CprE 281:
Digital Logic

Instructor: Alexander Stoytchev

http://www.ece.iastate.edu/~alexs/classes/

Intro to Verilog

Administrative Stuff

- **HW3 is due on Monday Sep 12 @ 10p**

Administrative Stuff

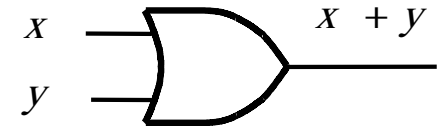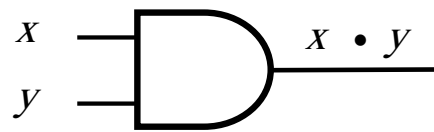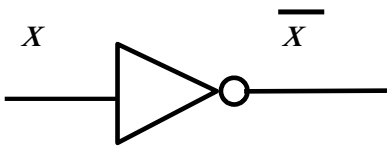- Sample homework solutions are posted on Canvas

- Look under 'Files'

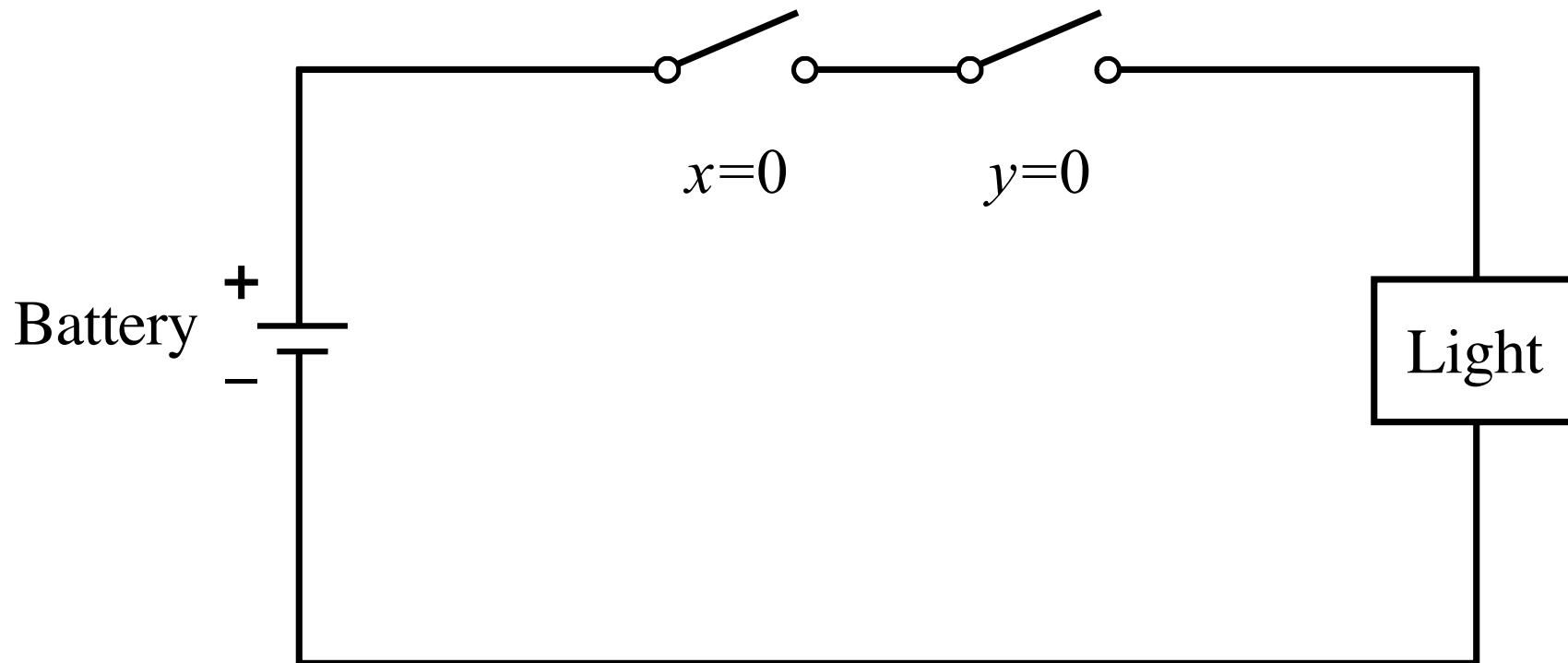# Quick Review

The Three Basic Logic Gates

x	NOT
0	1
1	0

x	y	AND
0	0	0
0	1	0
1	0	0
1	1	1

x	y	OR
0	0	0
0	1	1
1	0	1
1	1	1

x ⟶ $\overline{x}$

x, y ⟶ $x \cdot y$

x, y ⟶ $x + y$

AND with Switches

AND Circuit

Battery $+$ $-$

$x=0$ $y=0$

Light

AND Circuit

$x=0$ $y=1$

Battery

Light

AND Circuit

Battery

$+$

$-$

$x=1$ $y=0$

Light

AND Circuit

$x=1$ $y=1$

+

Battery

−
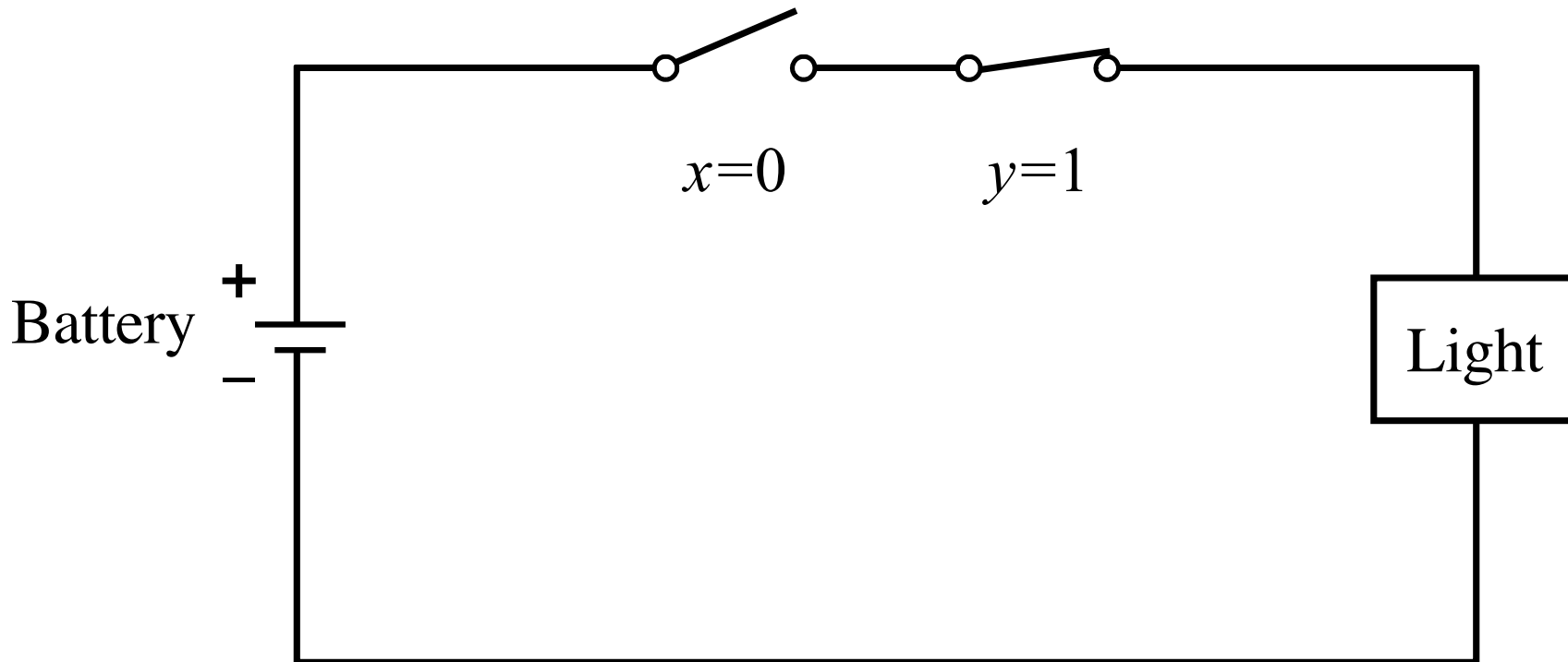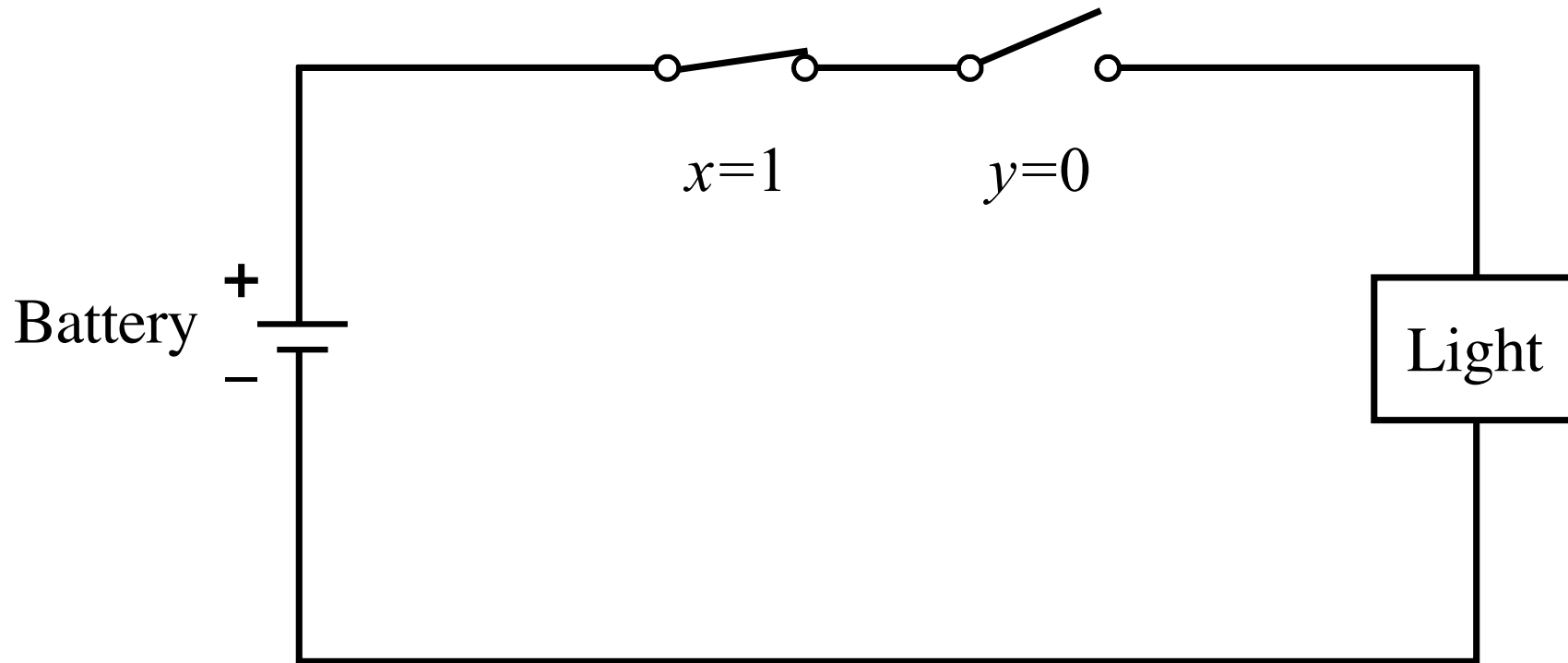
Light
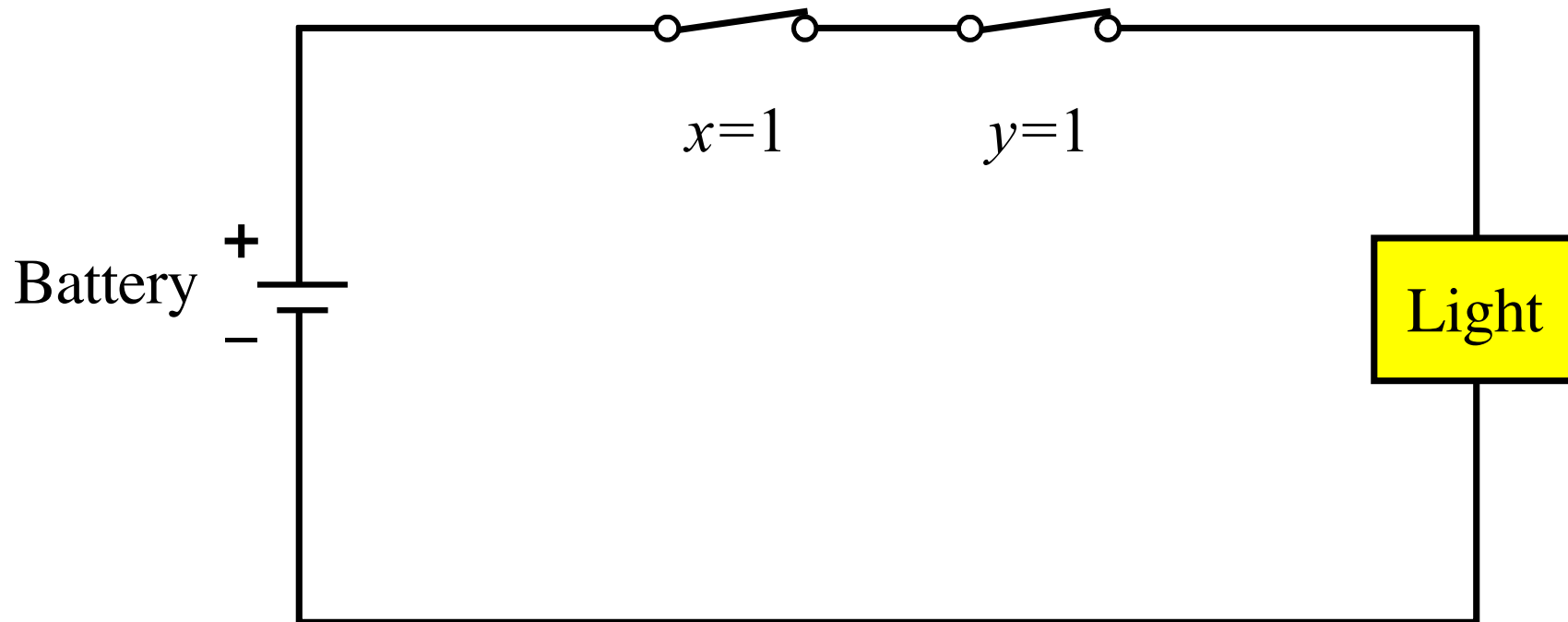
NOT with Switches

NOT Circuit

Resistor

Battery

$x=1$

Light

NOT Circuit

Resistor

Battery

+

−

$x=0$
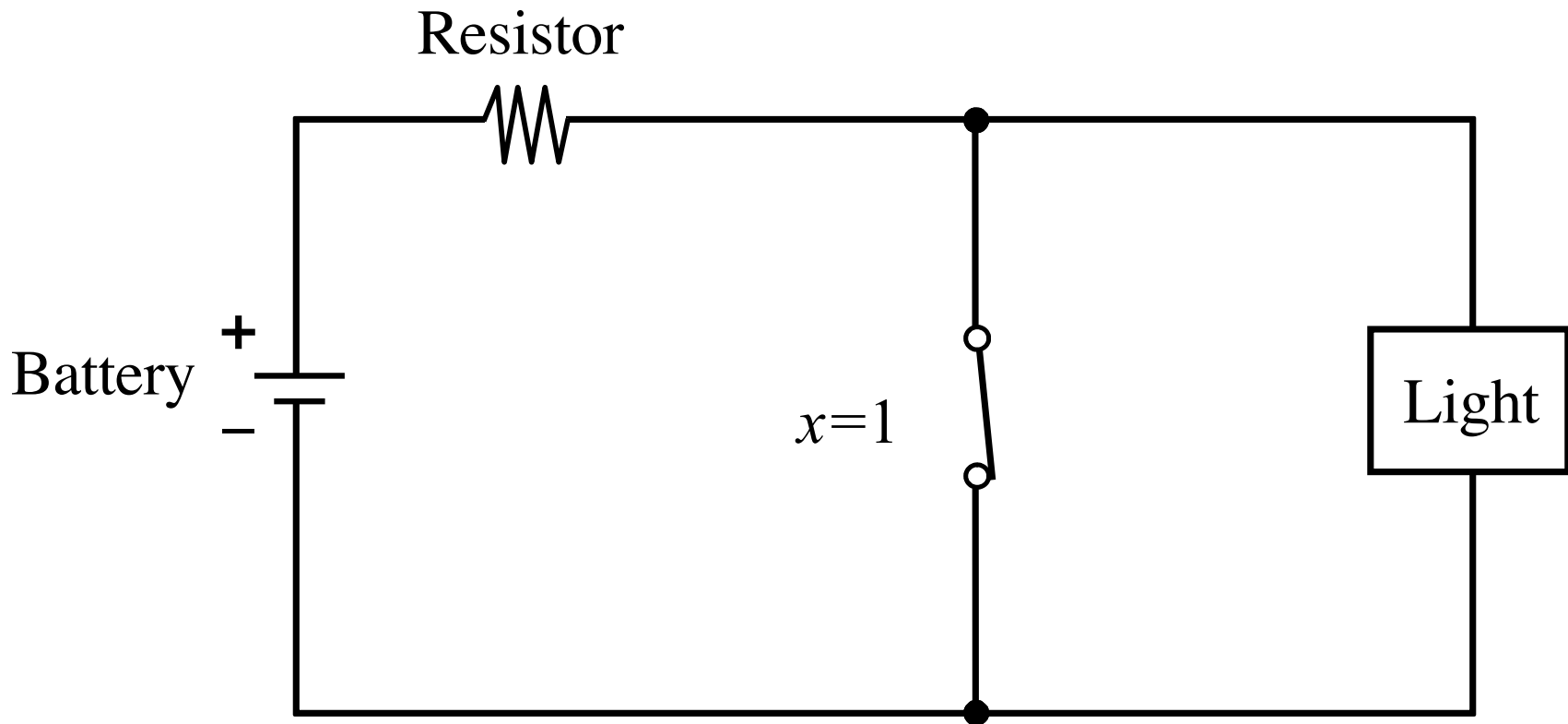
Light
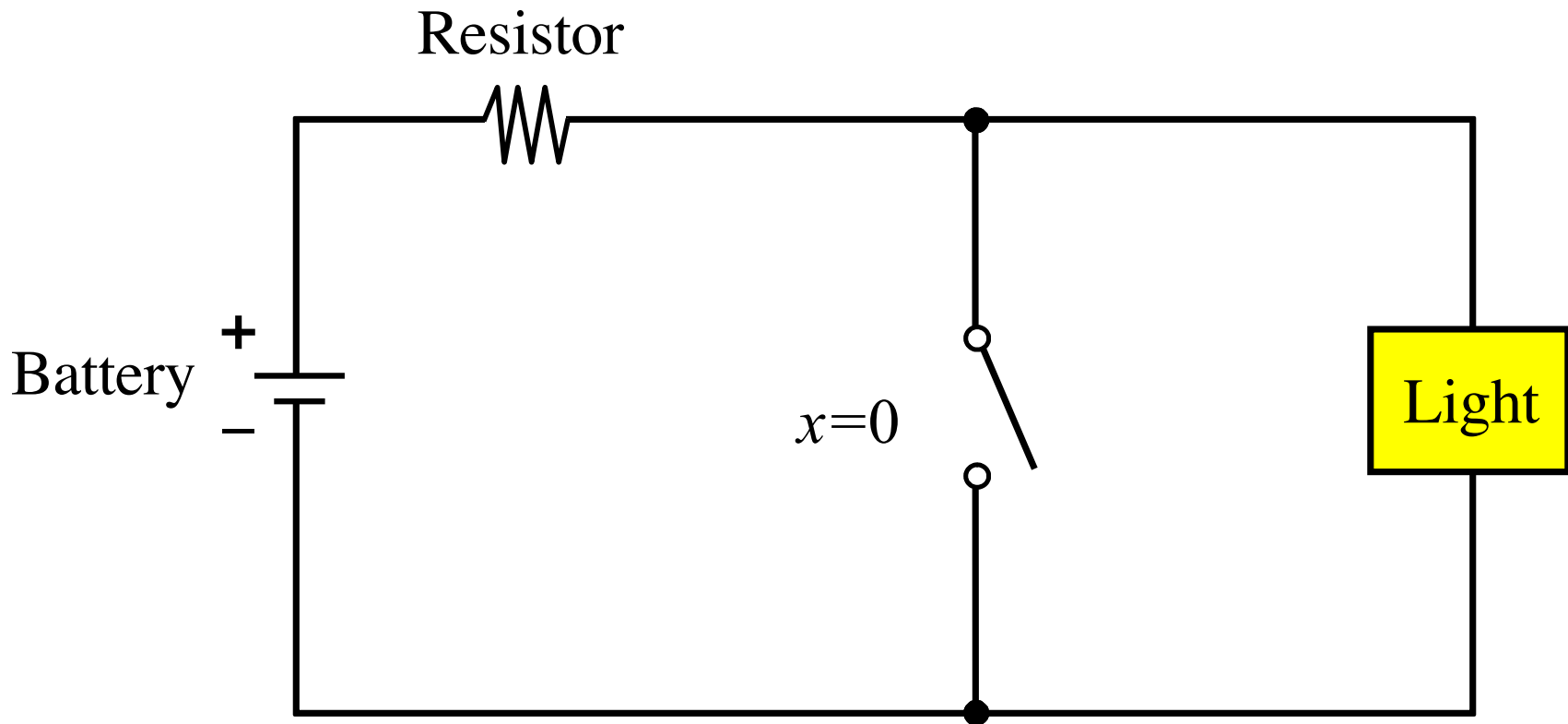
NAND with Switches

NAND Circuit

Resistor

Battery

+

−

$x=0$

$y=0$

Light
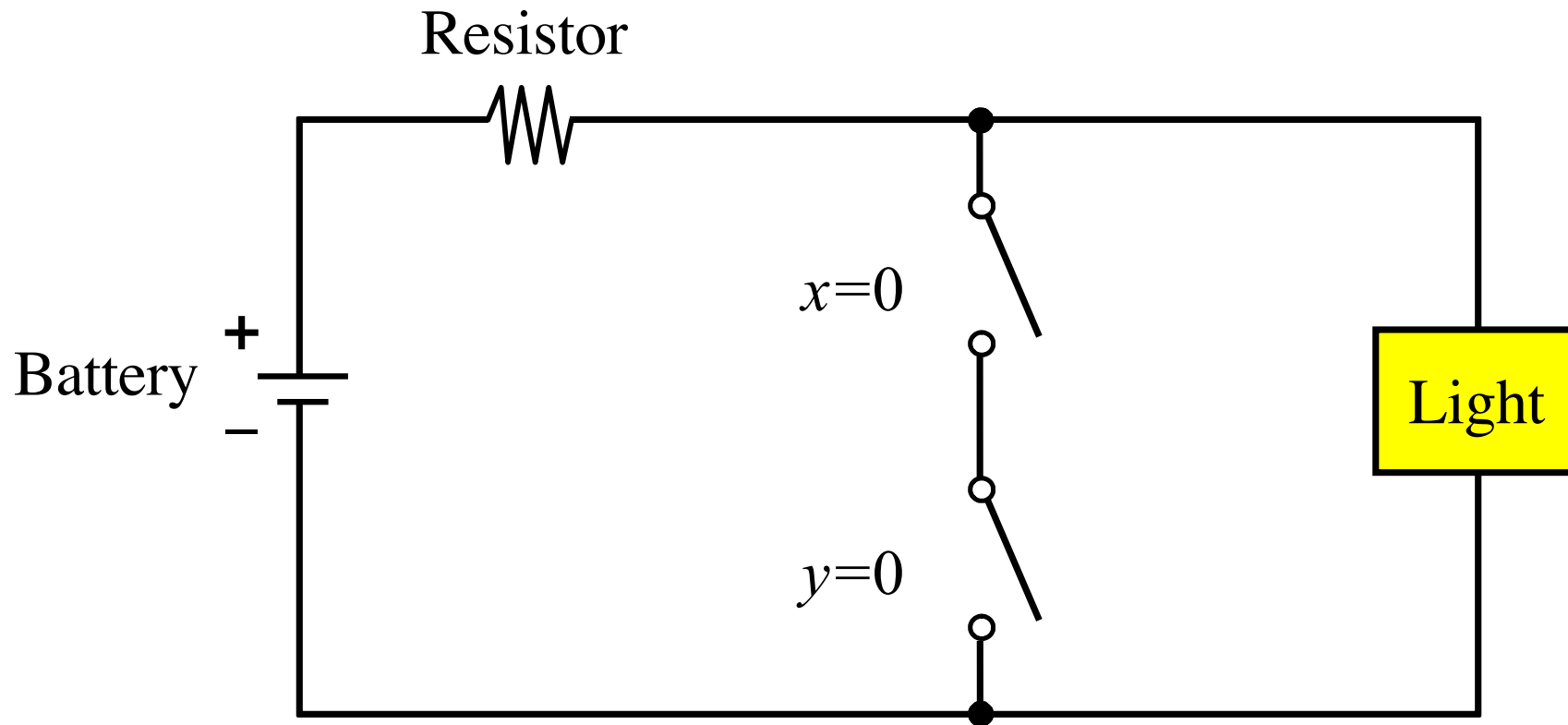
NAND Circuit

NAND Circuit

Resistor

Battery

$x=1$

$y=0$

Light

NAND Circuit

Resistor

Battery

$x=1$

$y=1$
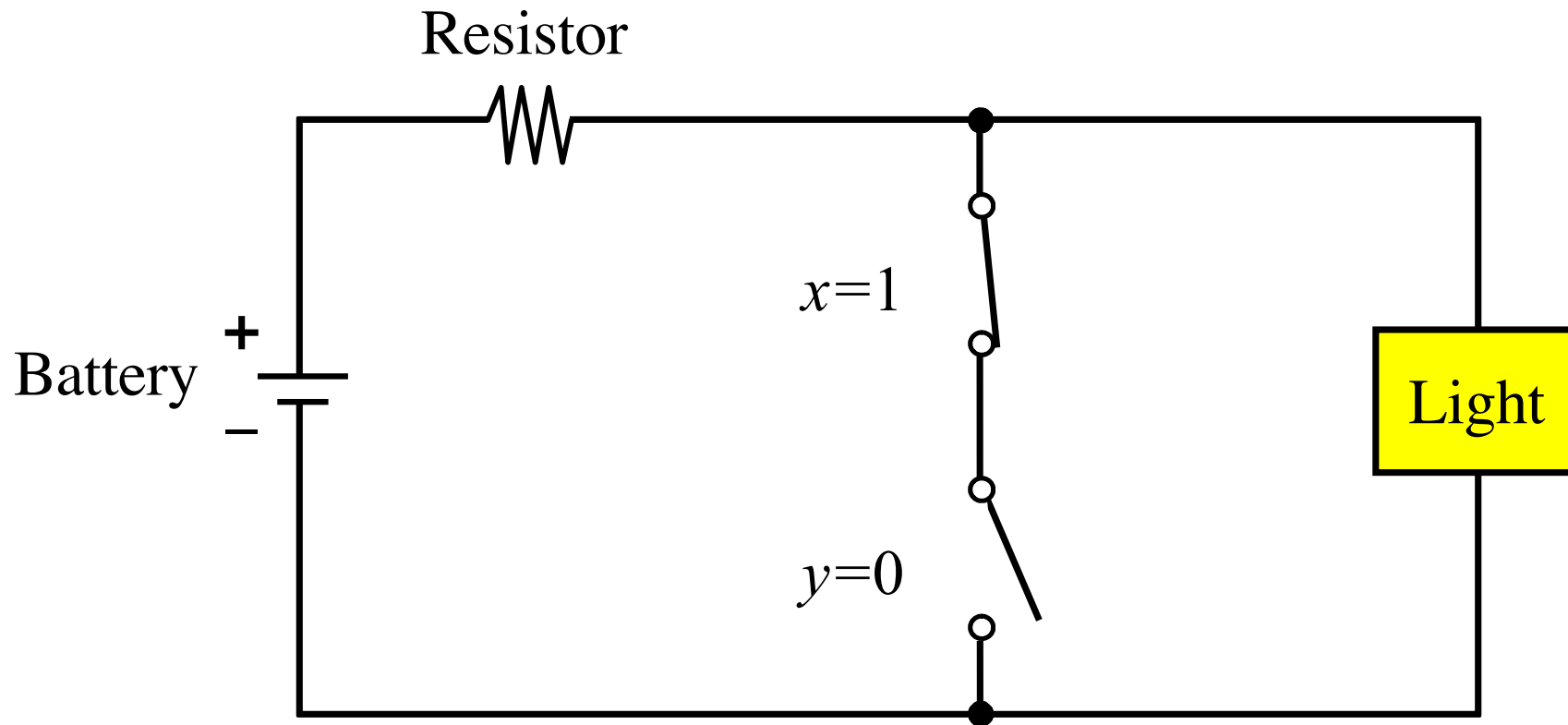
Light
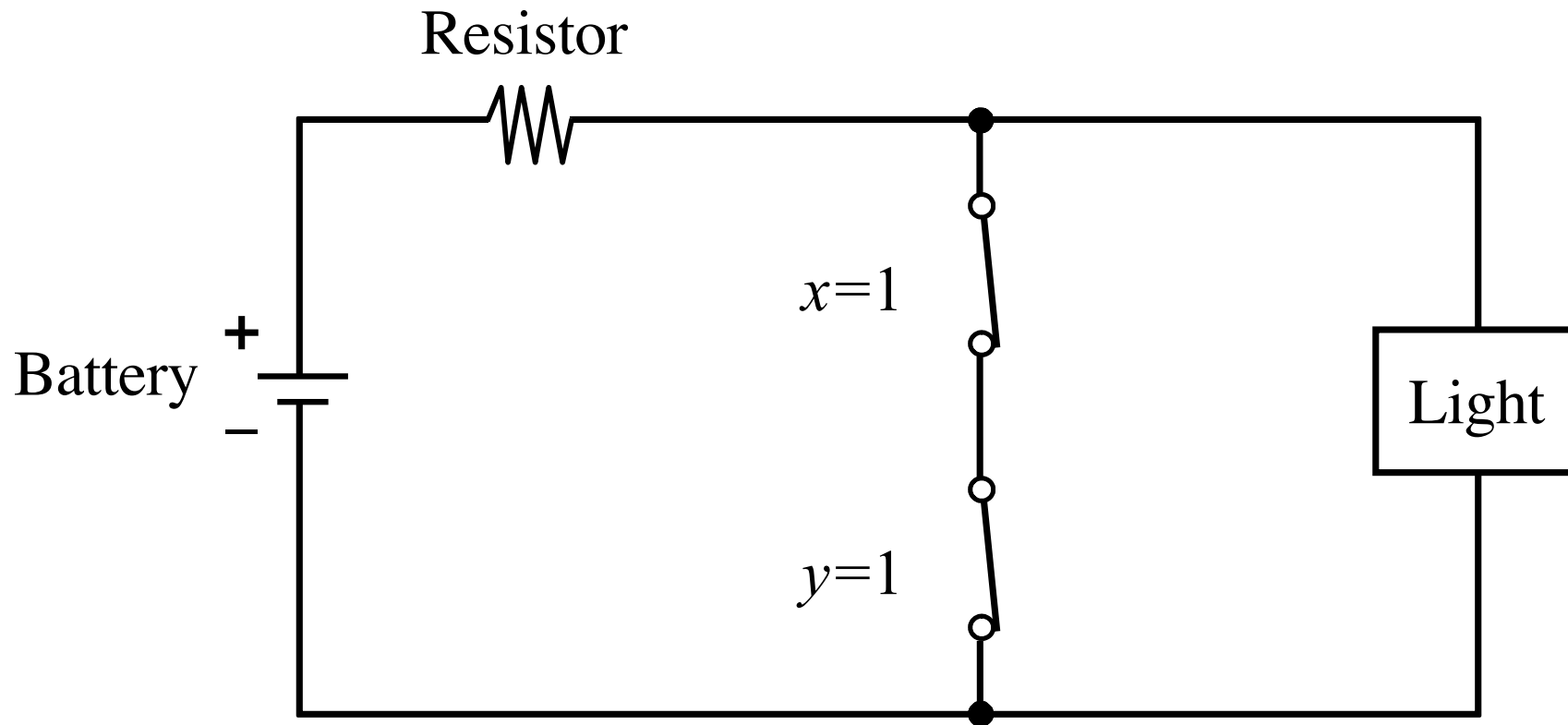
NOR with Switches

NOR Circuit
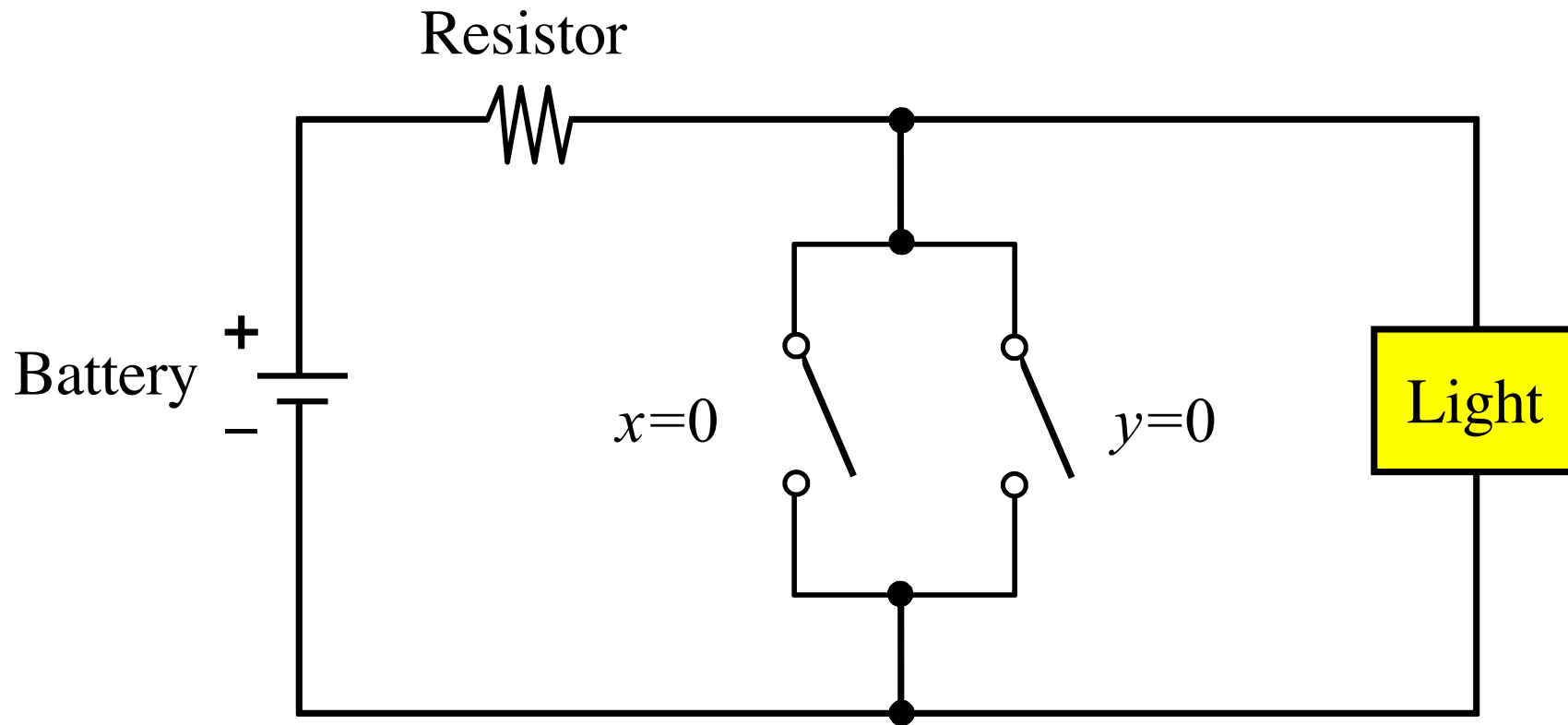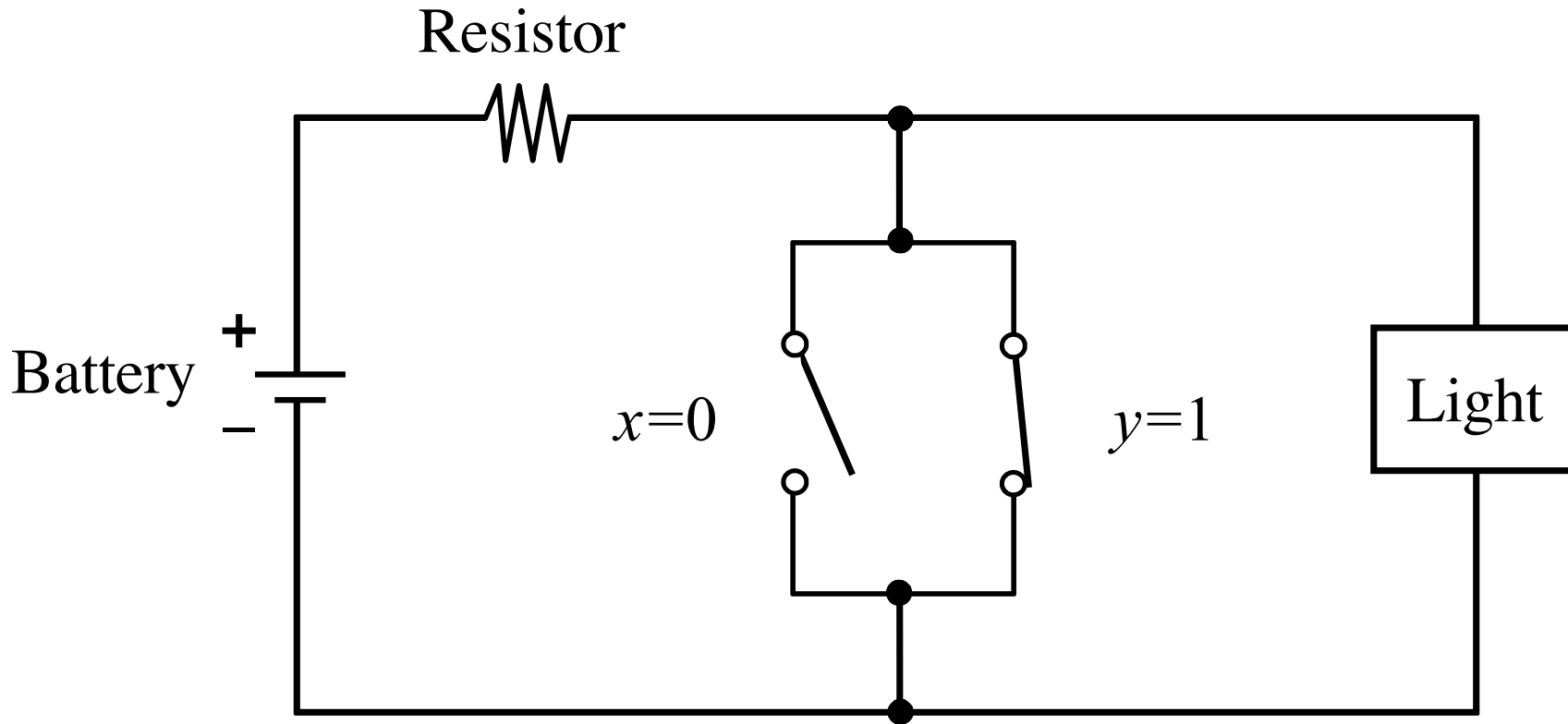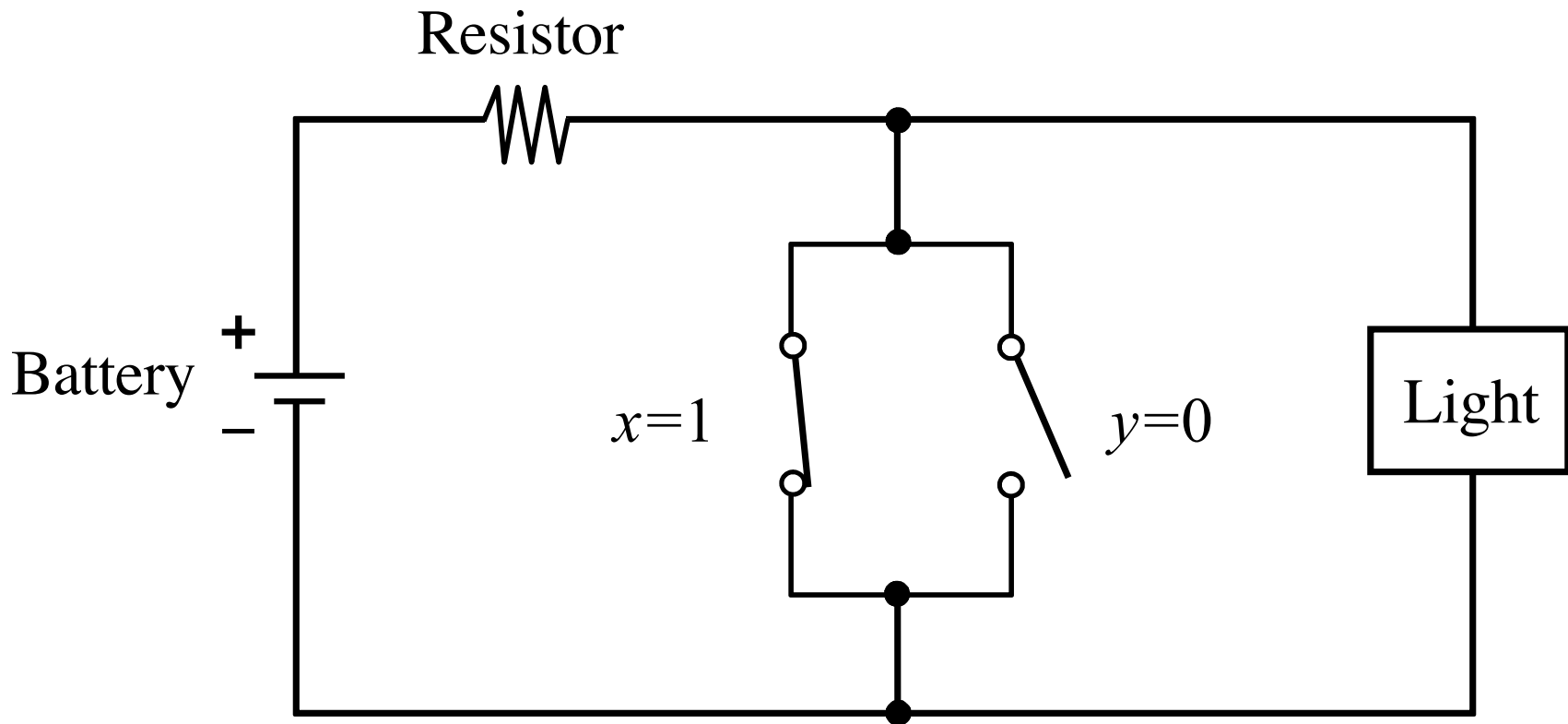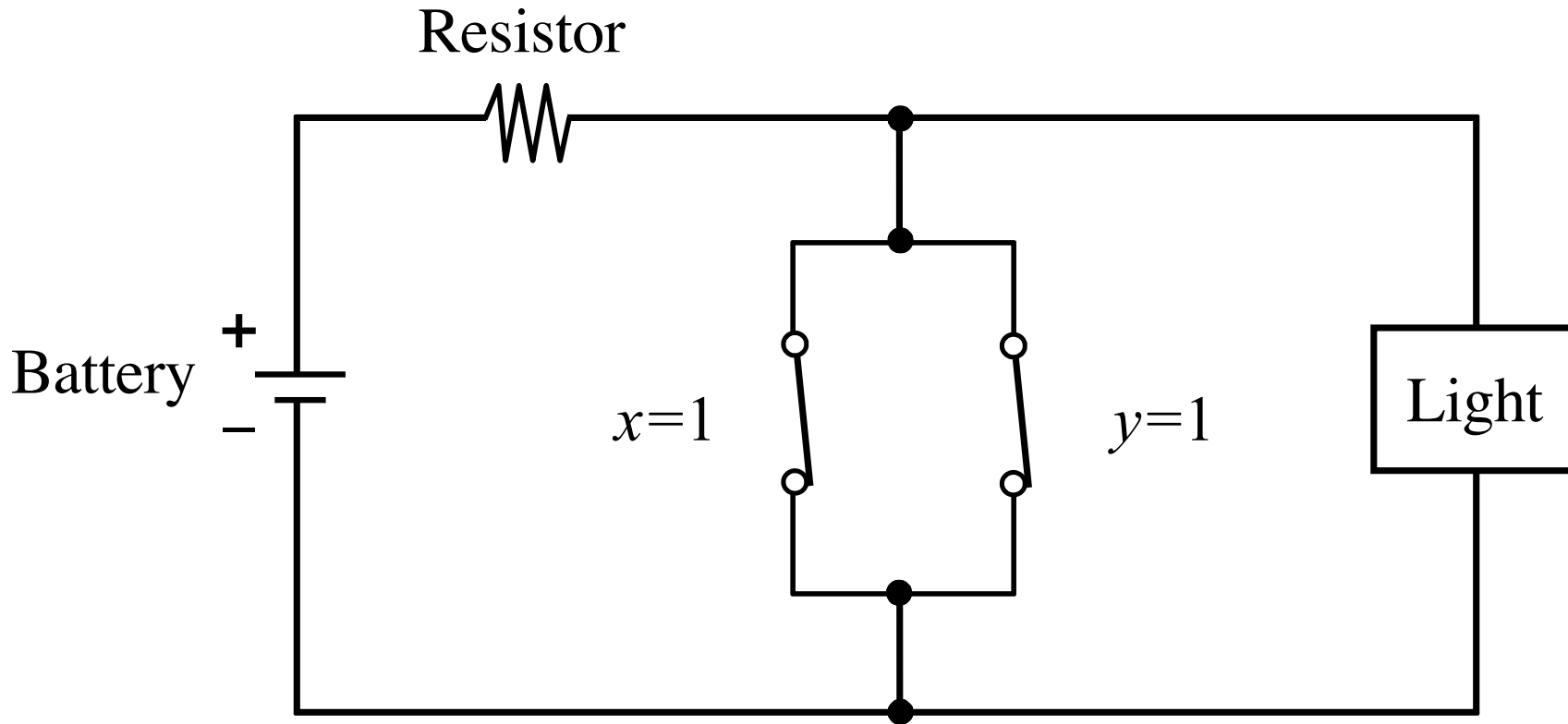
Resistor

Battery

+

−

$x=0$

$y=0$

Light

NOR Circuit

Resistor

Battery
+

−

$x=0$

$y=1$

Light

NOR Circuit

Resistor

Battery

+

−

x=1

y=0

Light

NOR Circuit

Resistor

Battery

+

−

$x=1$

$y=1$

Light

The Three Basic Logic Gates

x	NOT
0	1
1	0

x	y	AND
0	0	0
0	1	0
1	0	0
1	1	1

x	y	OR
0	0	0
0	1	1
1	0	1
1	1	1

x — NOT gate — $\overline{x}$

x, y — AND gate — $x \cdot y$

x, y — OR gate — $x + y$
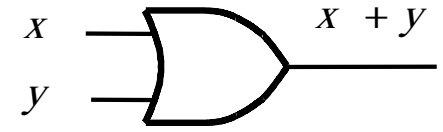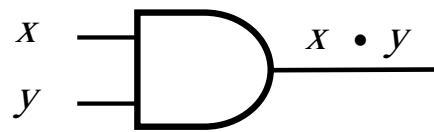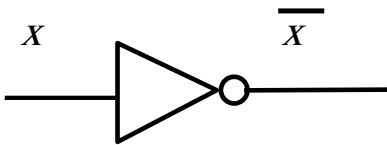
NAND and NOR

x	y	NAND
0	0	1
0	1	1
1	0	1
1	1	0

x	y	NOR
0	0	1
0	1	0
1	0	0
1	1	0

x

x

$\overline{x \cdot y}$

x

y

$\overline{x + y}$
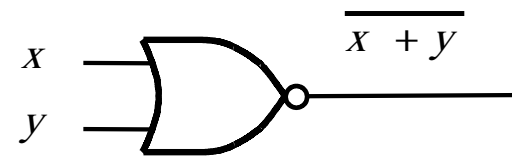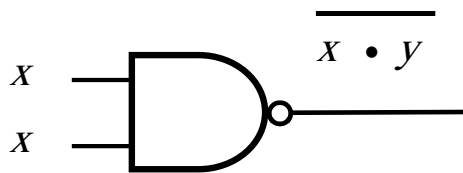
XOR and XNOR

x	y	XOR
0	0	0
0	1	1
1	0	1
1	1	0

x	y	XNOR
0	0	1
0	1	0
1	0	0
1	1	1

x ———

y ——— $x \oplus y$

x ———

y ——— $\overline{x \oplus y}$
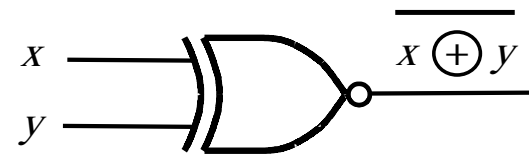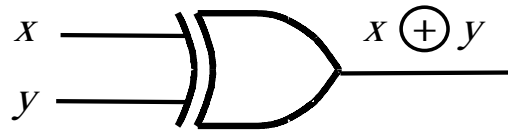
XNOR with Switches

XNOR Circuit

$x=0$

$y=0$

+

Battery

–

Light

XNOR Circuit

$x=0$

$y=1$

+

Battery

−

Light

XNOR Circuit

Battery

$x=1$

$y=0$

Light

XNOR Circuit

Battery

+

−

$x=1$

$y=1$
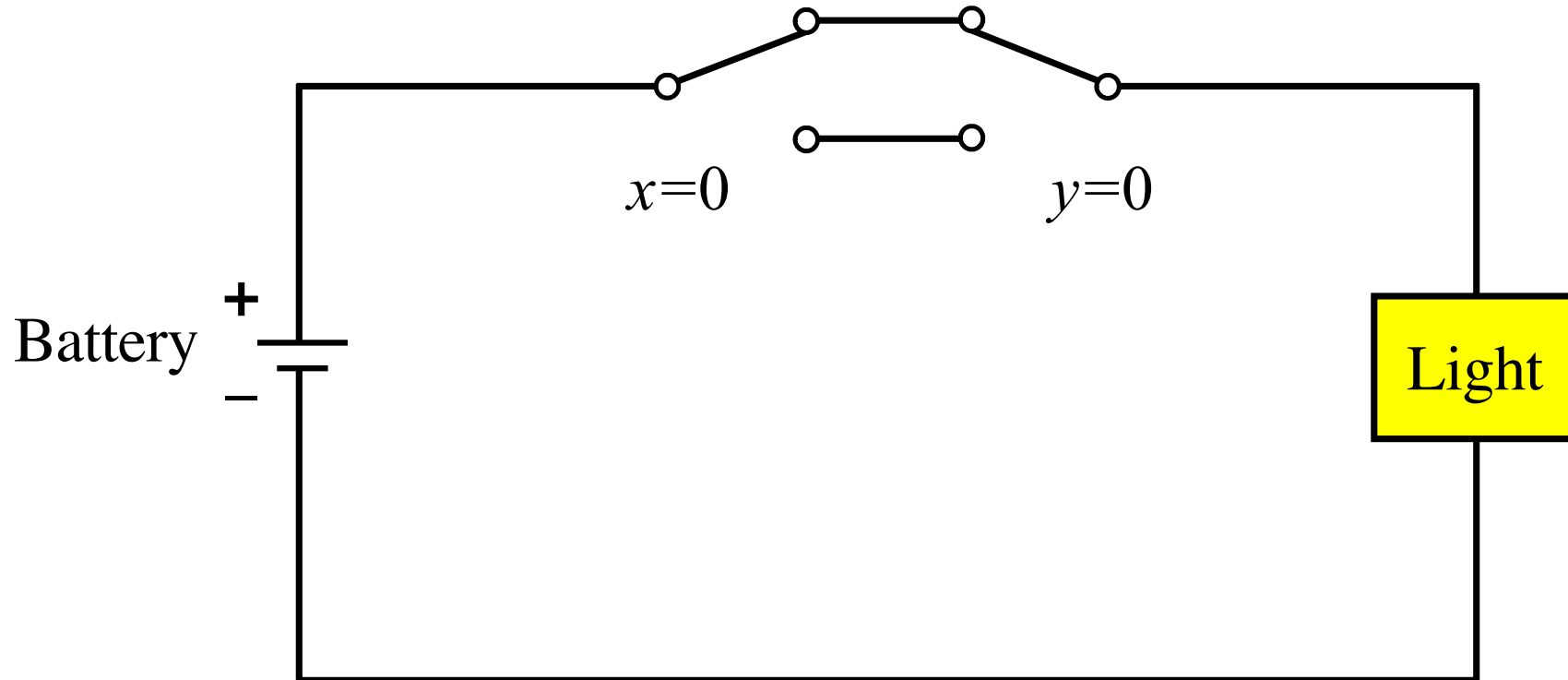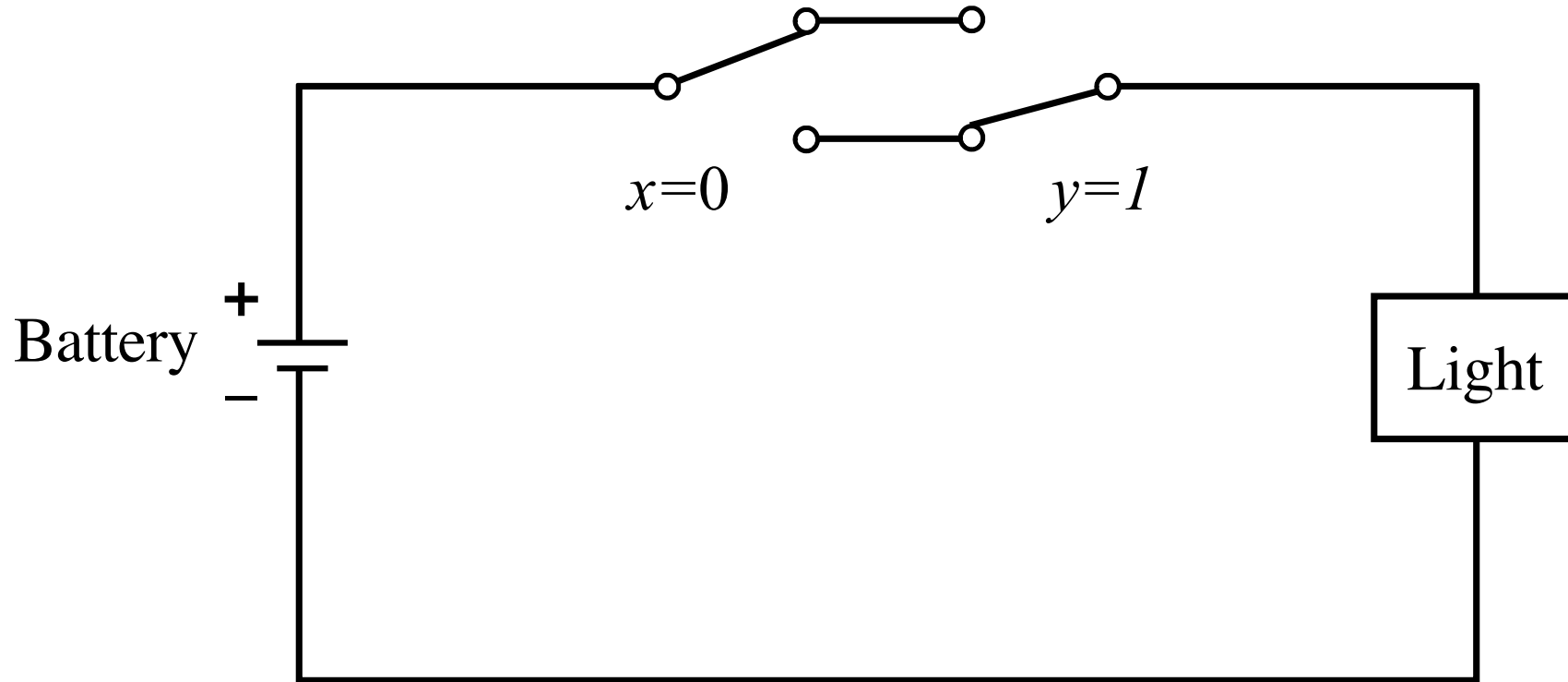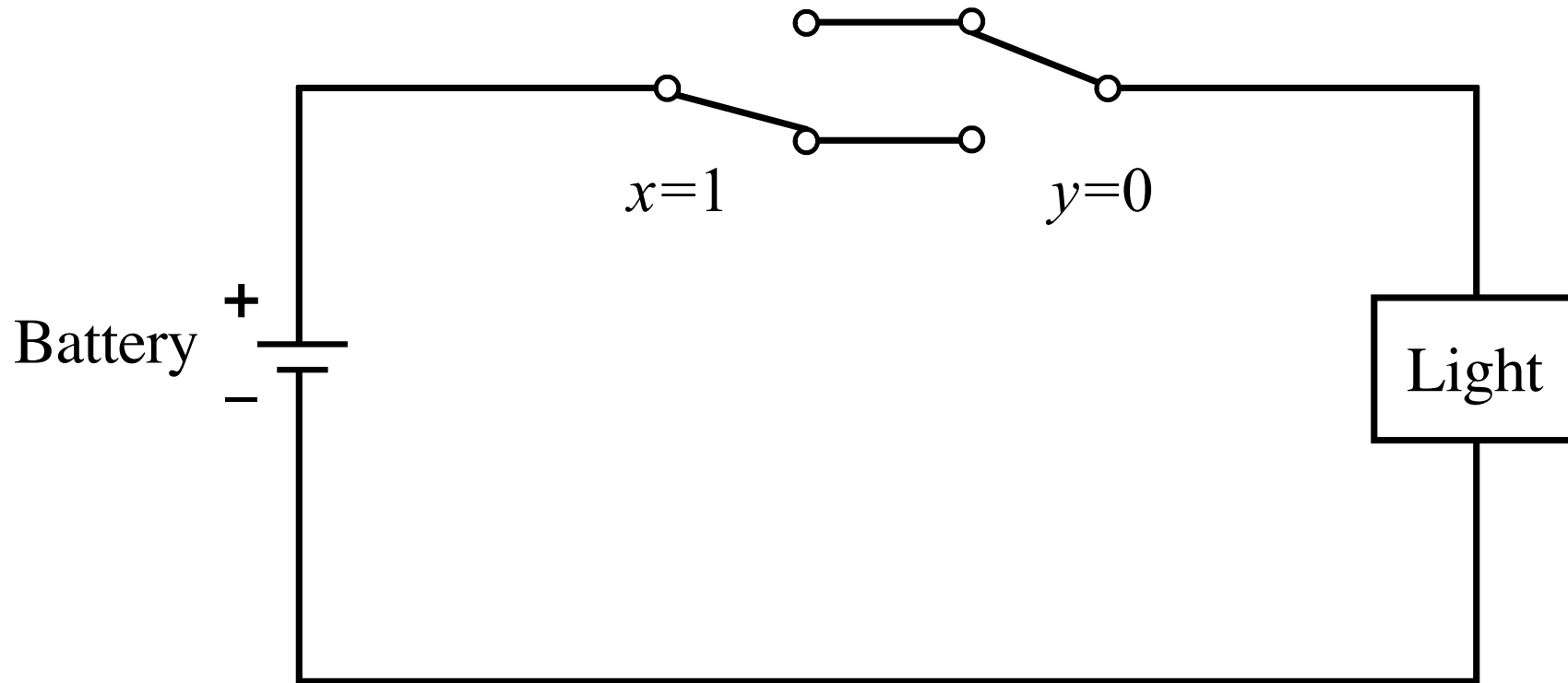
Light
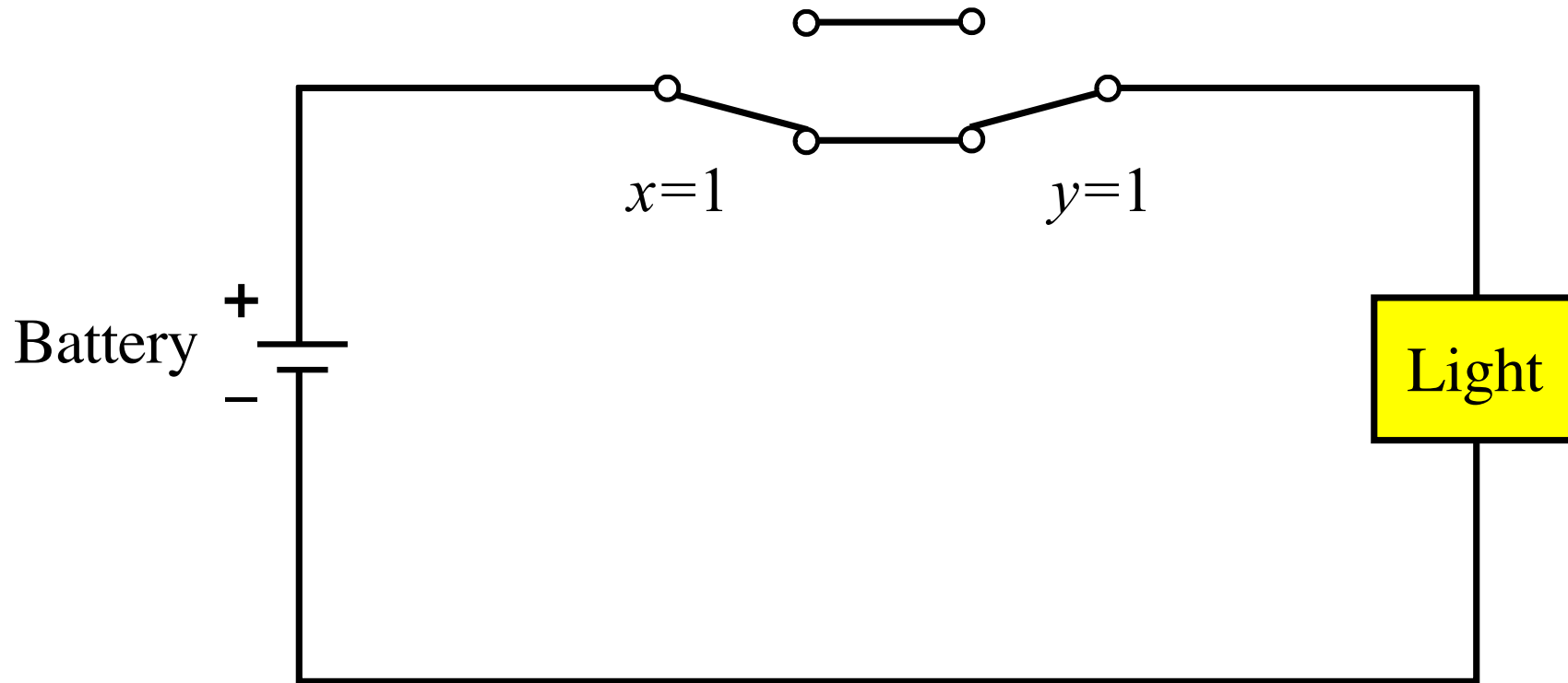
XOR with Switches

XOR Circuit

x=0   *y=0*

Battery   +   −   Light

XOR Circuit

$x=0$

$y=1$

+

Battery

−

Light

XOR Circuit

$x=1$

$y=0$

Battery

+

−

Light

XOR Circuit

$x=1$ $y=1$

Battery $+$

$-$
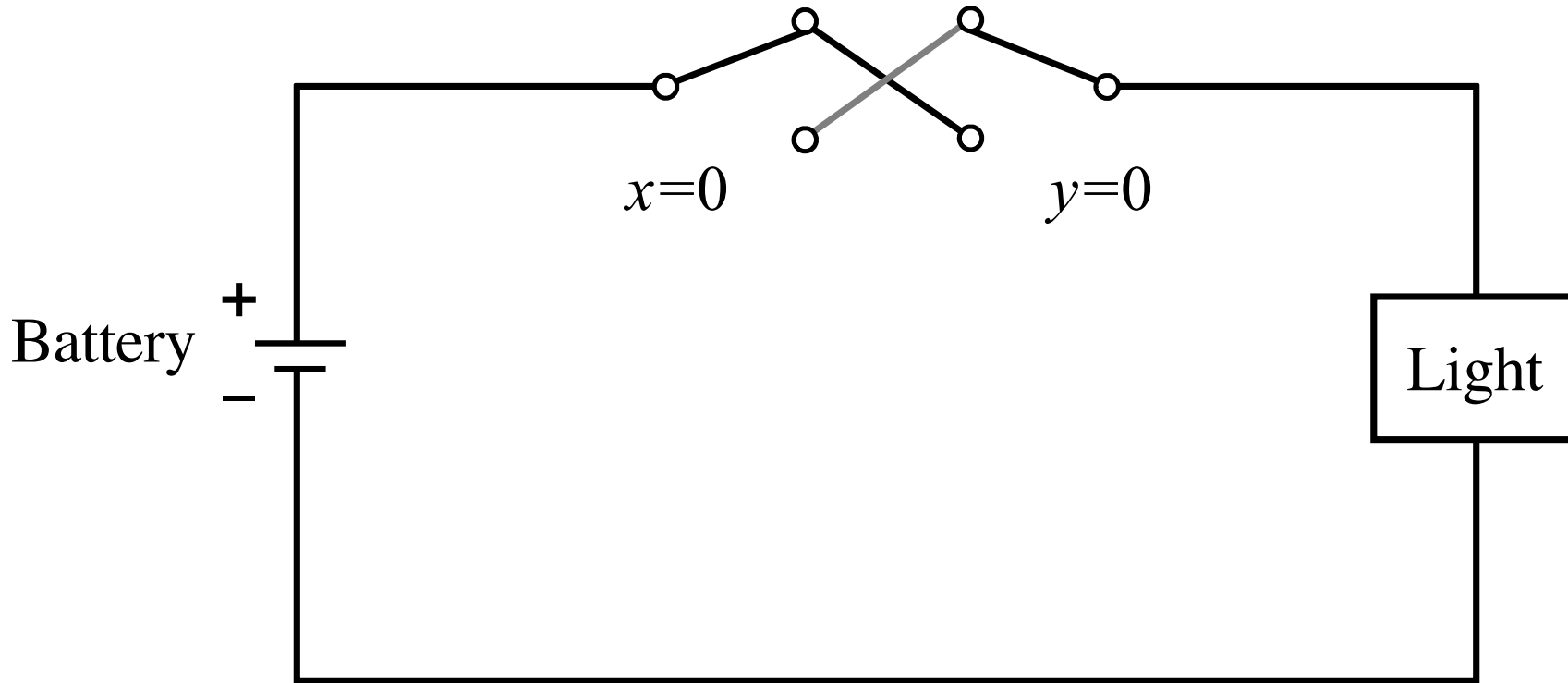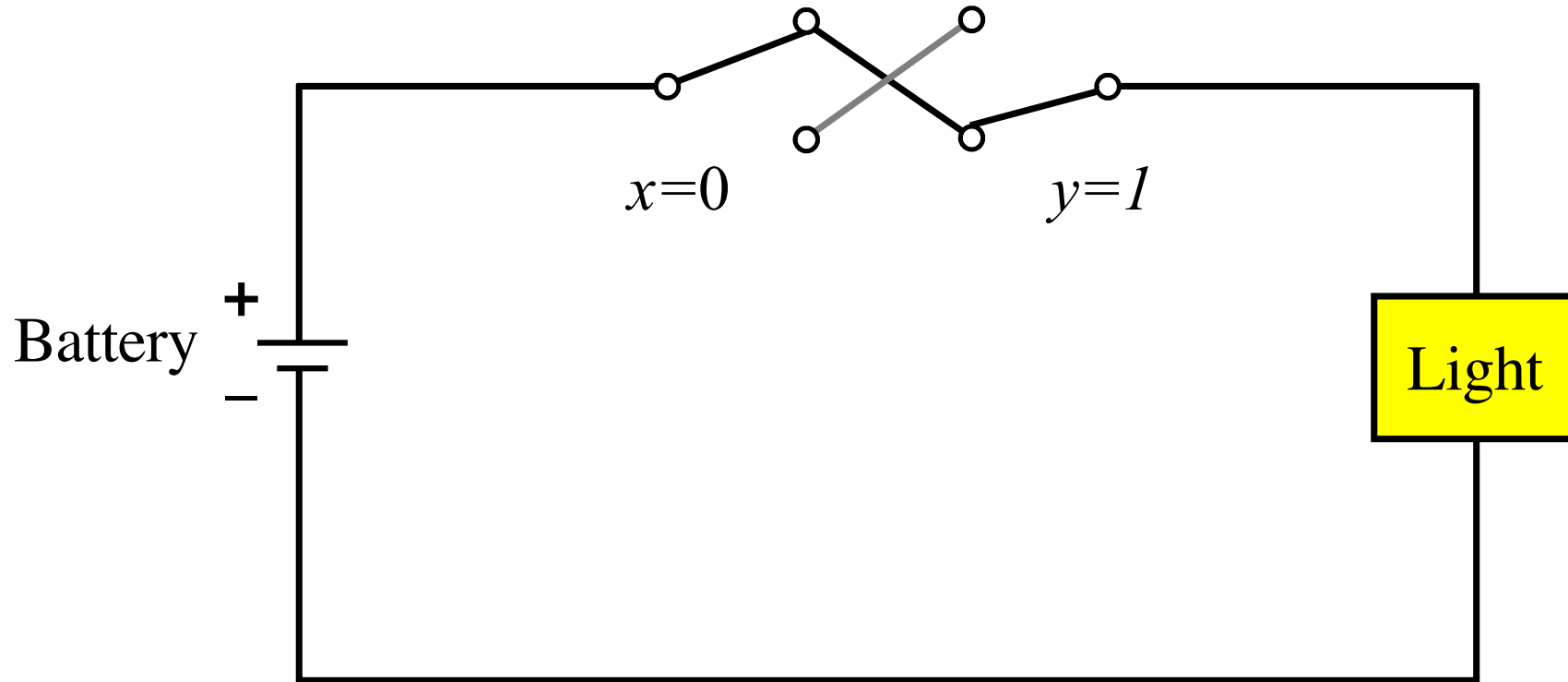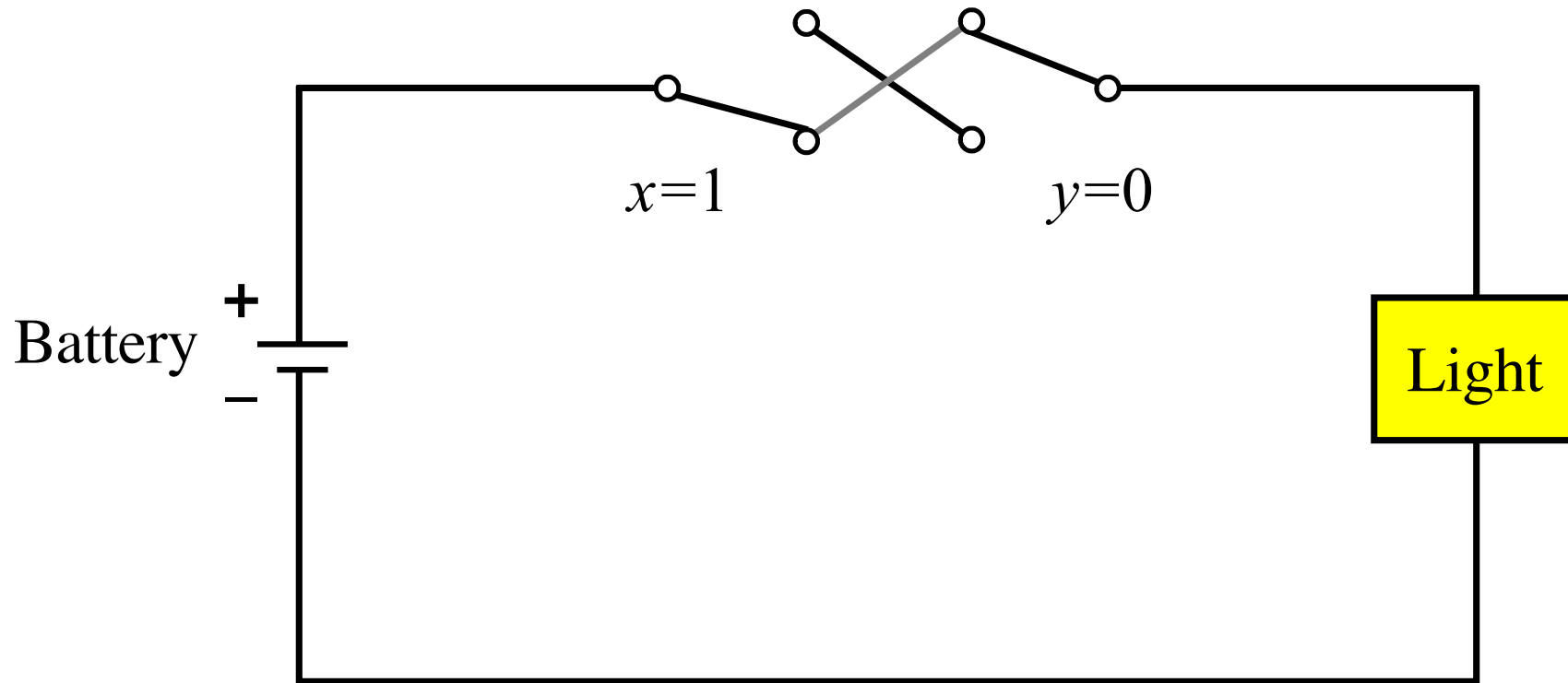
Light
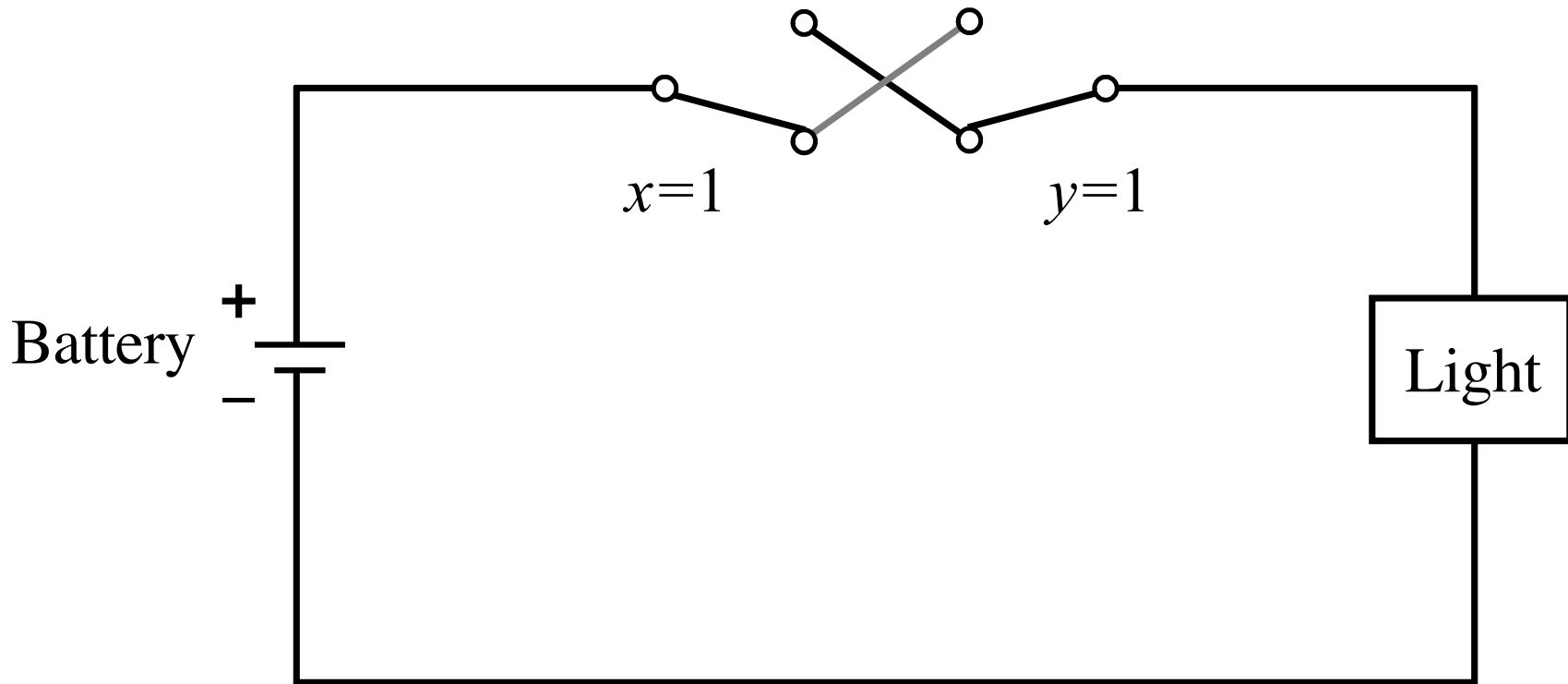
7-Segment Display Example

7-Segment Display
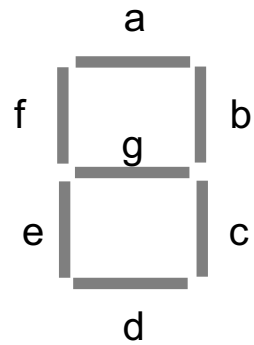
Displaying Some Numbers
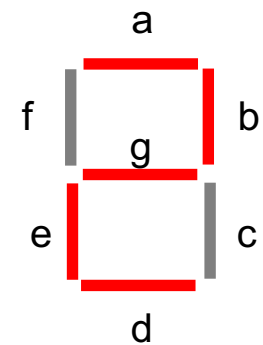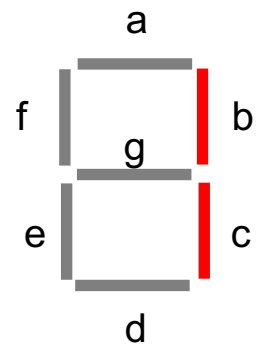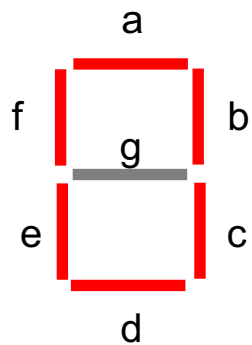
Displaying Some Hexadecimal Numbers
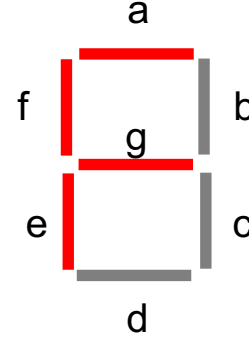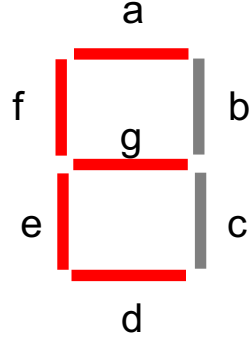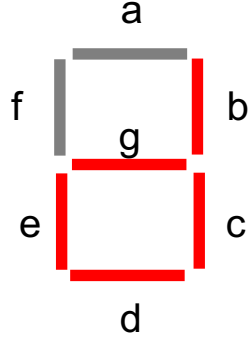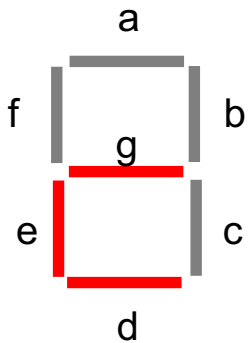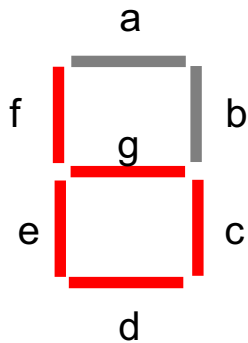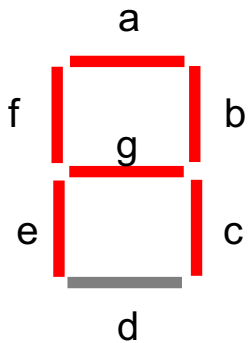
Display of numbers
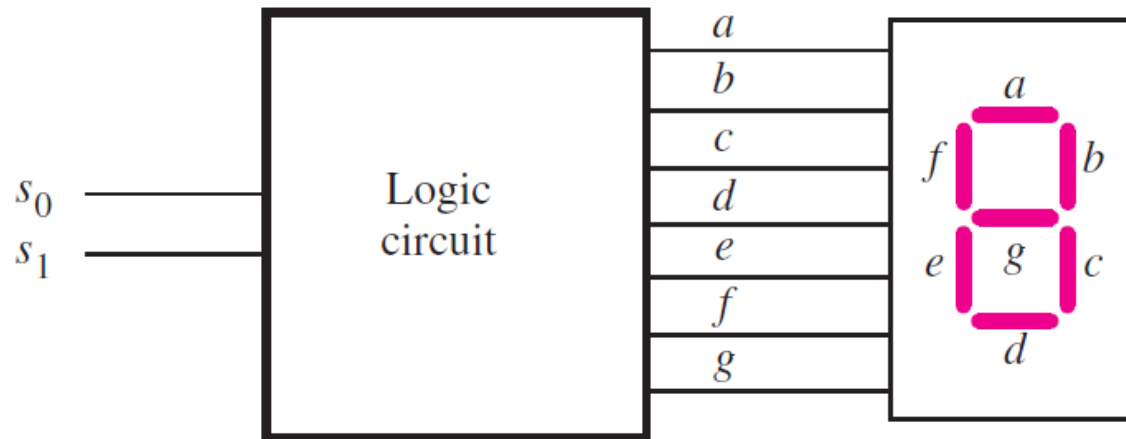
(a) Logic circuit and 7-segment display

s_1	s_0	a	b	c	d	e	f	g
0	0	1	1	1	1	1	1	0
0	1	0	1	1	0	0	0	0
1	0	1	1	0	1	1	0	1

(b) Truth table

[Figure 2.34 from the textbook]

Display of numbers

	s_1	s_0	a	b	c	d	e	f	g
0	0	0	1	1	1	1	1	1	0
1	0	1	0	1	1	0	0	0	0
2	1	0	1	1	0	1	1	0	1

Display of numbers

s_1	s_0	a	b	c	d	e	f	g
0	0	1	1	1	1	1	1	0
0	1	0	1	1	0	0	0	0
1	0	1	1	0	1	1	0	1

$$a = \overline{s_0} \qquad c = \overline{s_1} \qquad e = \overline{s_0} \qquad g = s_1\overline{s_0}$$

$$b = 1 \qquad d = \overline{s_0} \qquad f = \overline{s_1}\,\overline{s_0}$$

Intro to Verilog

History

- **Created in 1983/1984**

- **Verilog-95 (IEEE standard 1364-1995)**

- **Verilog 2001 (IEEE Standard 1364-2001)**

- **Verilog 2005 (IEEE Standard 1364-2005)**

- **SystemVerilog**

- **SystemVerilog 2009 (IEEE Standard 1800-2009).**

HDL

- **Hardware Description Language**

- **Verilog HDL**

- **VHDL**

Verilog HDL != VHDL

- **These are two different Languages!**

- **Verilog is closer to C**

- **VHDL is closer to Ada**
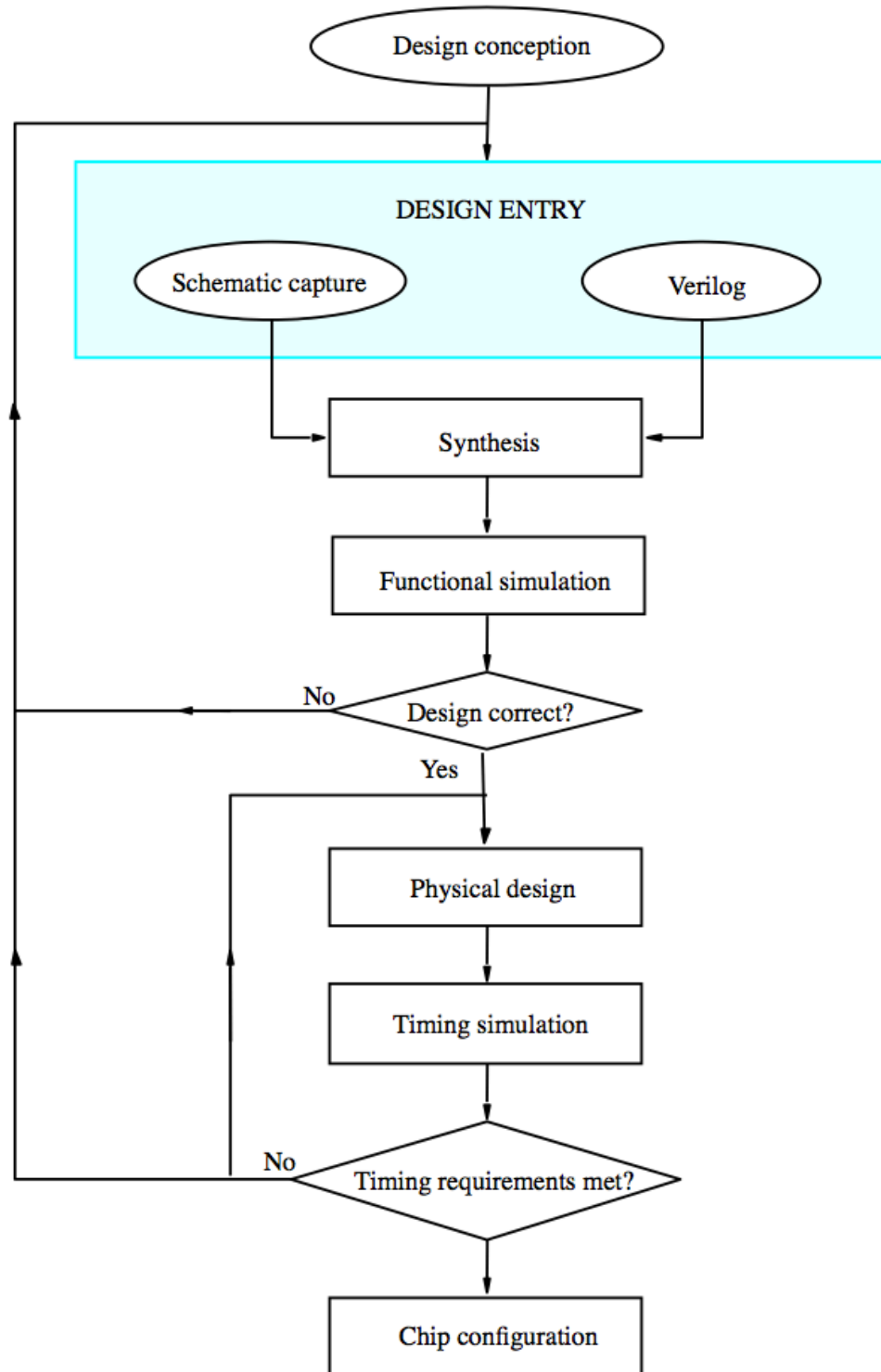
[Figure 2.35 from the textbook]

Sample Verilog Program

```
module example1 (x1, x2, s, f);
    input x1, x2, s;
    output f;

    not (k, s);
    and (g, k, x1);
    and (h, s, x2);
    or (f, g, h);

endmodule
```

The Three Basic Logic Gates

x —▷o— $\overline{x}$

x_1, x_2 — $x_1 \cdot x_2$

x_1, x_2 — $x_1 + x_2$
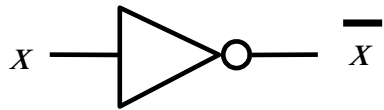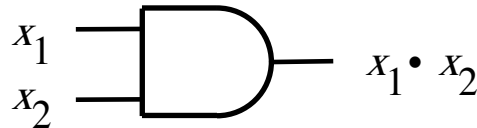
NOT gate

AND gate
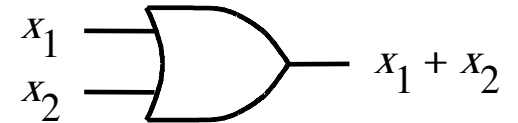
OR gate

You can build any circuit using only these three gates

[Figure 2.8 from the textbook]

How to specify a NOT gate in Verilog

X —▷o— $\overline{X}$

NOT gate

How to specify a NOT gate in Verilog

we'll use the letter y for the output

x ——▷o—— y

NOT gate

How to specify a NOT gate in Verilog

x —▷○— y
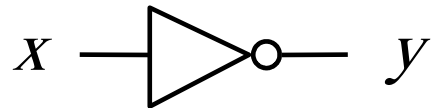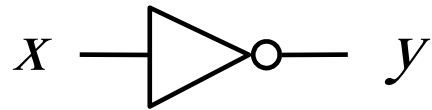
not (y, x)

NOT gate

Verilog code

How to specify an AND gate in Verilog

x_1

x_2
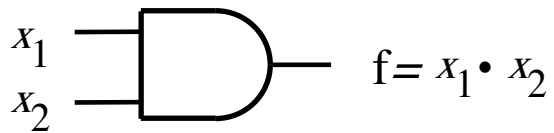
$f = x_1 \cdot x_2$

and $(f, x1, x2)$

AND gate

Verilog code

How to specify an OR gate in Verilog

x_1
x_2
$f = x_1 + x_2$
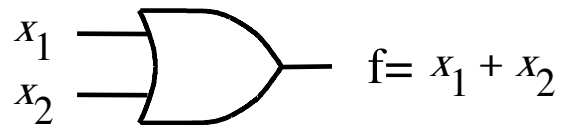
or (f, x1, x2)

OR gate

Verilog code

Verilog Code

Structural   Behavioral
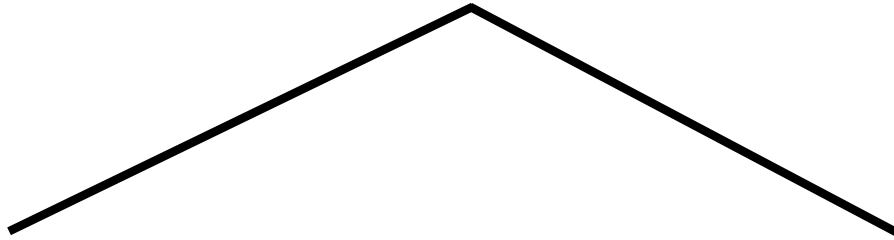
Structural Verilog

2-to-1 Multiplexer in Verilog (structural syntax)

2-1 Multiplexer

[Figure 2.36 from the textbook]

Verilog Code for a 2-1 Multiplexer


```
module example1 (x1, x2, s, f);
    input  x1, x2, s;
    output  f;

    not (k, s);
    and (g, k, x1);
    and (h, s, x2);
    or (f, g, h);

endmodule
```

[Figure 2.36 from the textbook]   [Figure 2.37 from the textbook]

Verilog Code for a 2-1 Multiplexer


```
module  example1 (x1, x2, s, f);
    input  x1, x2, s;
    output  f;

    not (k, s);
    and (g, k, x1);
    and (h, s, x2);
    or (f, g, h);

endmodule
```

[Figure 2.36 from the textbook]

[Figure 2.37 from the textbook]

Verilog Code for a 2-1 Multiplexer

[Figure 2.36 from the textbook]

```
module example1 (x1, x2, s, f);
    input x1, x2, s;
    output f;

    not (k, s);
    and (g, k, x1);
    and (h, s, x2);
    or (f, g, h);

endmodule
```

[Figure 2.37 from the textbook]

Verilog Code for a 2-1 Multiplexer

[Figure 2.36 from the textbook]

```
module example1 (x1, x2, s, f);
    input  x1, x2, s;
    output  f;

    not (k, s);
    and (g, k, x1);
    and (h, s, x2);
    or (f, g, h);

endmodule
```

[Figure 2.37 from the textbook]

Verilog Code for a 2-1 Multiplexer

x_1

k

s

x_2

f

```
module example1 (x1, x2, s, f);
    input  x1, x2, s;
    output  f;

    not (k, s);
    and (g, k, x1);
    and (h, s, x2);
    or (f, g, h);

endmodule
```

[Figure 2.36 from the textbook]   [Figure 2.37 from the textbook]

Verilog Code for a 2-1 Multiplexer


```
module example1 (x1, x2, s, f);
    input  x1, x2, s;
    output  f;

    not (k, s);
    and (g, k, x1);
    and (h, s, x2);
    or (f, g, h);

endmodule
```

[Figure 2.36 from the textbook]   [Figure 2.37 from the textbook]

Verilog Code for a 2-1 Multiplexer


```
module example1 (x1, x2, s, f);
    input  x1, x2, s;
    output  f;

    not (k, s);
    and (g, k, x1);
    and (h, s, x2);
    or (f, g, h);

endmodule
```
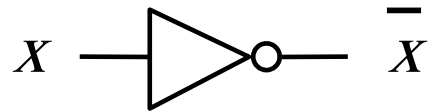
[Figure 2.36 from the textbook]   [Figure 2.37 from the textbook]

Verilog Code for a 2-1 Multiplexer


```
module example1 (x1, x2, s, f);
    input  x1, x2, s;
    output  f;

    not (k, s);
    and (g, k, x1);
    and (h, s, x2);
    or (f, g, h);

endmodule
```

[Figure 2.36 from the textbook]   [Figure 2.37 from the textbook]

Verilog Code for a 2-1 Multiplexer


```
module example1 (x1, x2, s, f);
    input  x1, x2, s;
    output  f;

    not (k, s);
    and (g, k, x1);
    and (h, s, x2);
    or (f, g, h);

endmodule
```

[Figure 2.36 from the textbook]

[Figure 2.37 from the textbook]

Verilog Code for a 2-1 Multiplexer


```
module example1 (x1, x2, s, f);
    input  x1, x2, s;
    output  f;

    not (k, s);
    and (g, k, x1);
    and (h, s, x2);
    or (f, g, h);

endmodule
```

[Figure 2.36 from the textbook]   [Figure 2.37 from the textbook]

Verilog Code for a 2-1 Multiplexer


```
module example1 (x1, x2, s, f);
    input  x1, x2, s;
    output  f;

    not (k, s);
    and (g, k, x1);
    and (h, s, x2);
    or (f, g, h);

endmodule
```

[Figure 2.36 from the textbook]   [Figure 2.37 from the textbook]
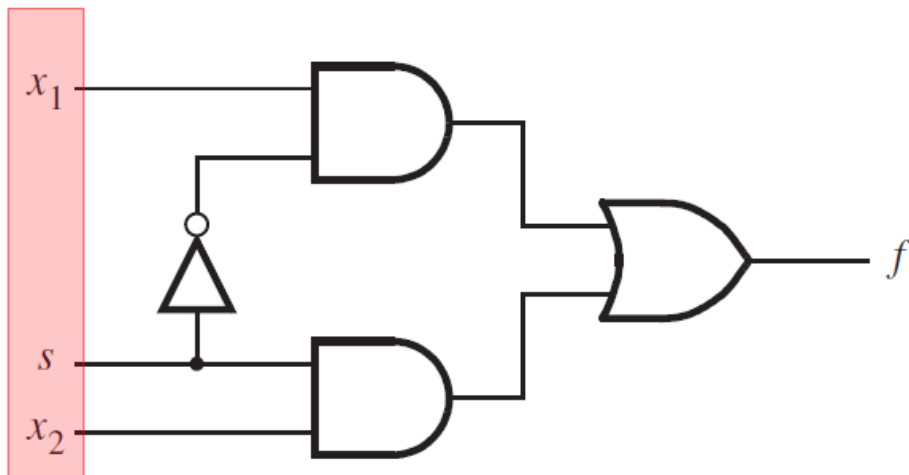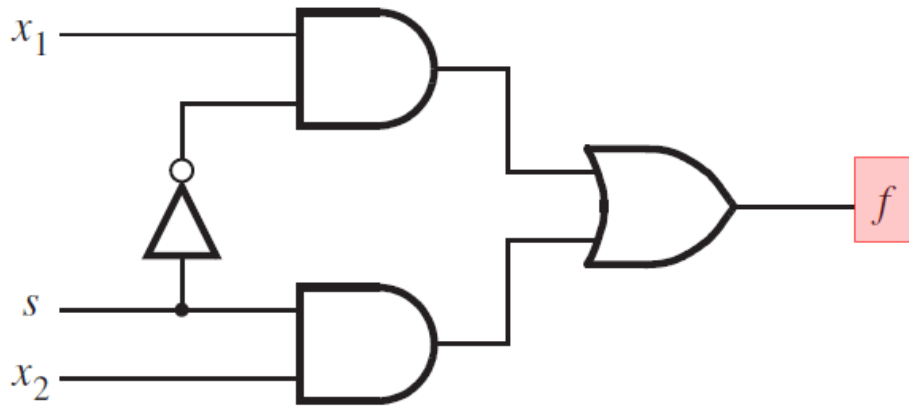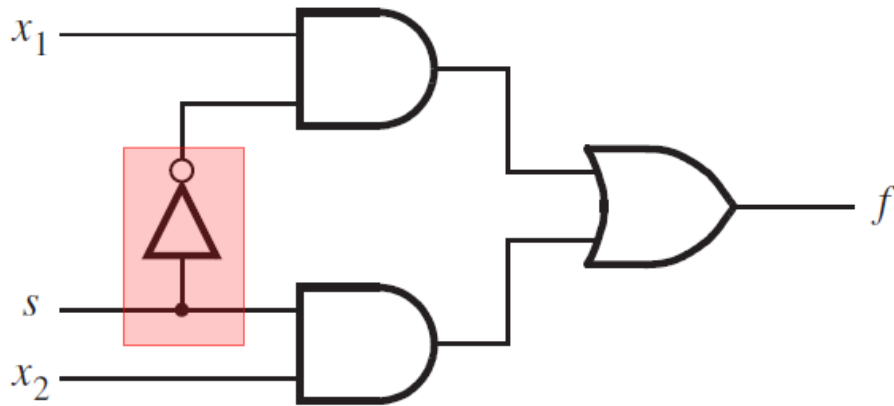
Behavioral Verilog

Verilog Code
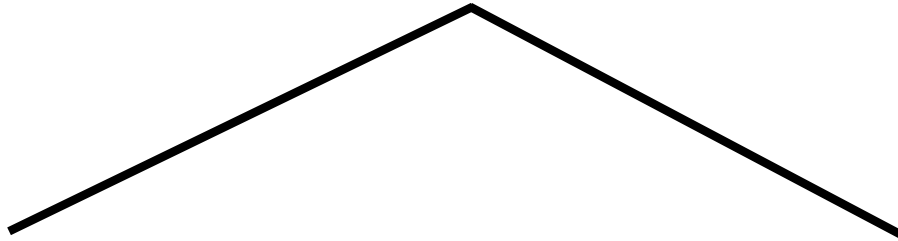
Structural   Behavioral
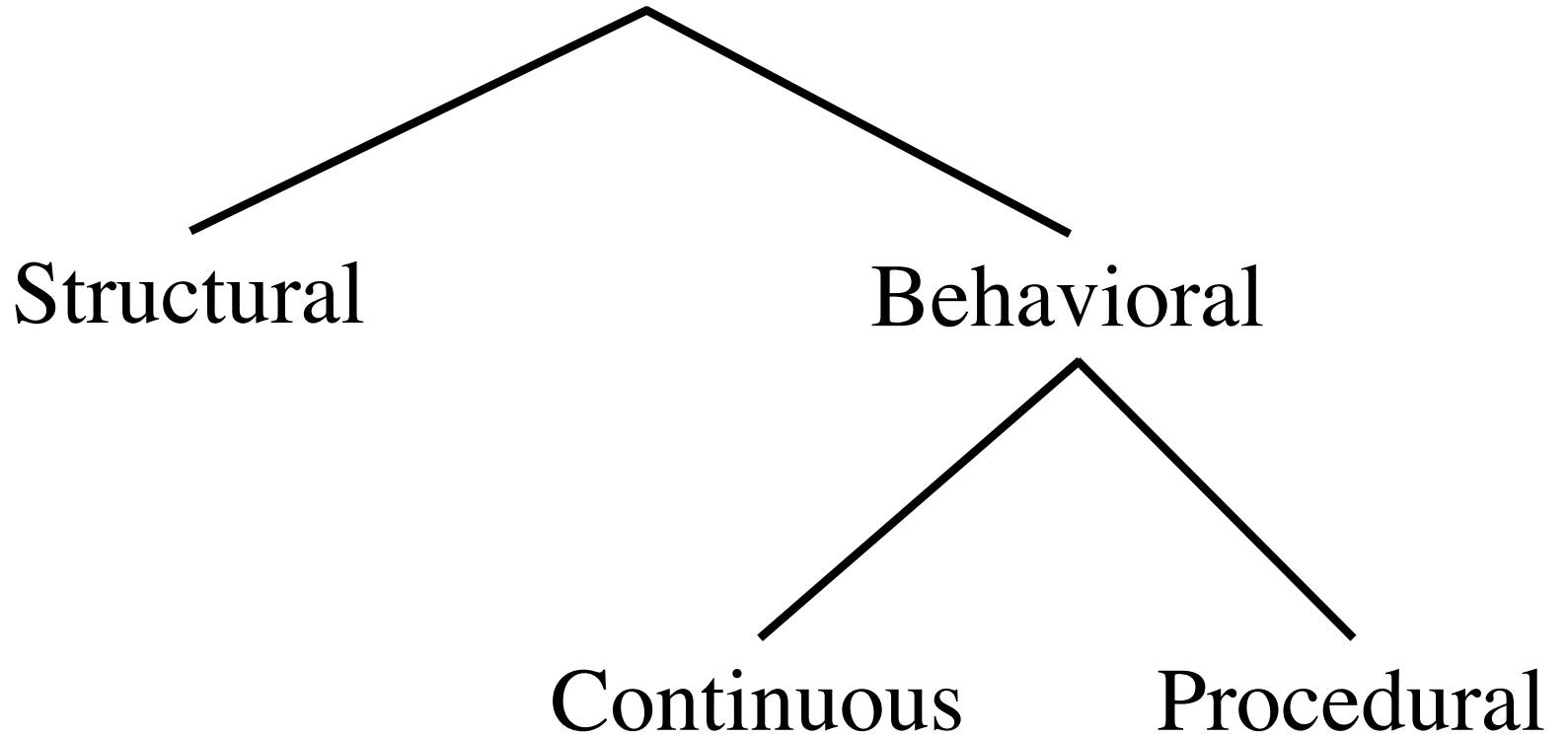
2-to-1 Multiplexer in Verilog (behavioral-continuous syntax)

Verilog Code for a 2-1 Multiplexer

// Behavioral-Continuous specification

module example3 (x1, x2, s, f);
    input x1, x2, s;
    output f;

    assign f = (~s & x1) | (s & x2);

endmodule

[Figure 2.36 from the textbook]   [Figure 2.40 from the textbook]

Verilog Code for a 2-1 Multiplexer

[Figure 2.36 from the textbook]

// Behavioral-Continuous specification

module example3 (x1, x2, s, f);
   input x1, x2, s;
   output f;

   assign f = ($\sim$s & x1) | (s & x2);

endmodule

[Figure 2.40 from the textbook]

Verilog Code for a 2-1 Multiplexer

// Behavioral-Continuous specification

module example3 (x1, x2, s, f);
    input x1, x2, s;
    output f;

    assign f = (~s & x1) | (s & x2);

endmodule

[Figure 2.36 from the textbook]

[Figure 2.40 from the textbook]

Verilog Code for a 2-1 Multiplexer

// Behavioral-Continuous specification

module example3 (x1, x2, s, f);
    input x1, x2, s;
    output f;

    assign f = (~s & x1) | (s & x2);

endmodule

[Figure 2.36 from the textbook]

[Figure 2.40 from the textbook]
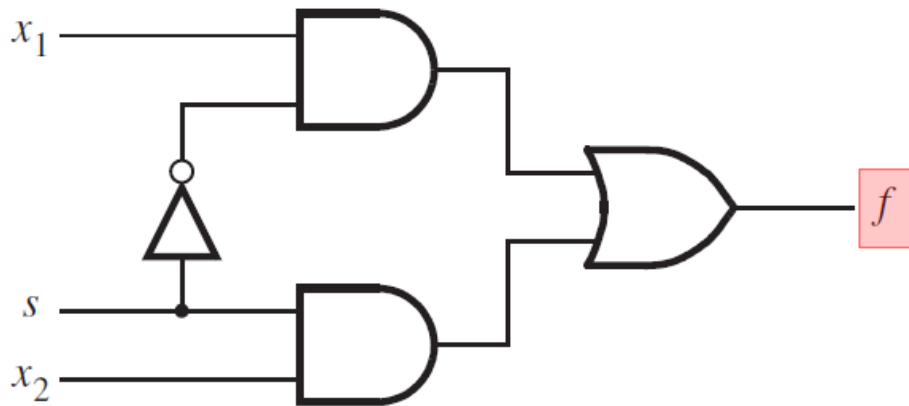
2-to-1 Multiplexor in Verilog (behavioral-procedural specification)

Verilog Code for a 2-1 Multiplexer

// Behavioral-Procedural specification

```
module  example5 (x1, x2, s, f);
        input  x1, x2, s;
        output f;
        reg f;

        always @(x1 or x2 or s)
                if (s == 0)
                        f = x1;
                else
                        f = x2;

endmodule
```

[Figure 2.36 from the textbook]   [Figure 2.42 from the textbook]

Verilog Code for a 2-1 Multiplexer

A 2-to-1 Multiplexer has:

- **one output f**

- **two inputs: x_1 and x_2**

- **It also has another input line s**

- **If s=0, then the output is equal to x_1**

- **If s=1, then the output is equal to x_2**

// Behavioral-Procedural specification

```
module example5 (x1, x2, s, f);
    input  x1, x2, s;
    output f;
    reg f;

    always @(x1 or x2 or s)
        if (s == 0)
            f = x1;
        else
            f = x2;

endmodule
```

[Figure 2.36 from the textbook]

[Figure 2.42 from the textbook]

Verilog Code for a 2-1 Multiplexer

A 2-to-1 Multiplexer has:

- **one output f**

- **two inputs: x_1 and x_2**

- **It also has another input line s**

- **If s=0, then the output is equal to x_1**

- **If s=1, then the output is equal to x_2**

// Behavioral-Procedural specification

```verilog
module example5 (x1, x2, s, f);
    input  x1, x2, s;
    output f;
    reg f;

    always @(x1 or x2 or s)
        if (s == 0)
            f = x1;
        else
            f = x2;

endmodule
```

[Figure 2.36 from the textbook]

[Figure 2.42 from the textbook]

always@ syntax

always @(x,y)

always @(x or y)

always @(*)

always @*

Verilog Code for a 2-1 Multiplexer
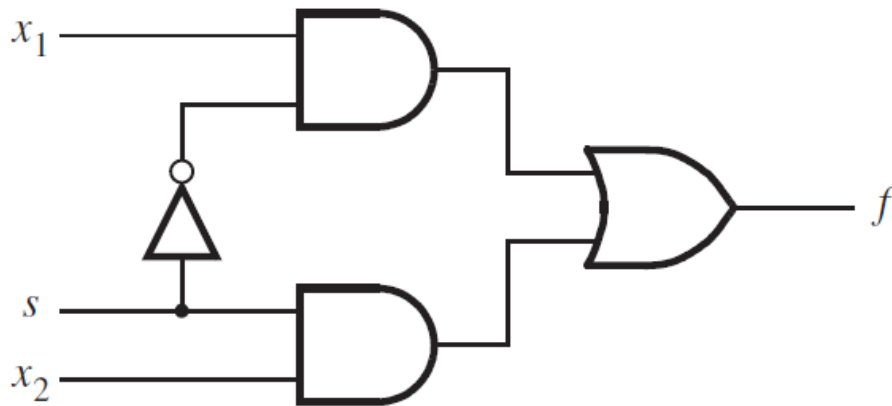
// Behavioral-Procedural specification

```
module  example5 (input x1, x2, s, output reg f);

    always @(x1, x2, s)
        if (s == 0)
            f = x1;
        else
            f = x2;

endmodule
```

[Figure 2.36 from the textbook]   [Figure 2.43 from the textbook]

Structural v.s. Behavioral

Verilog Code:
Structural v.s. Behavioral

// Structural specification

```
module  example1 (x1, x2, s, f);
    input  x1, x2, s;
    output  f;

    not (k, s);
    and (g, k, x1);
    and (h, s, x2);
    or (f, g, h);

endmodule
```

// Behavioral-Continuous specification

```
module  example3 (x1, x2, s, f);
    input x1, x2, s;
    output f;

    assign  f = (~s & x1) | (s & x2);

endmodule
```

[Figure 2.37 from the textbook]

[Figure 2.40 from the textbook]

Verilog Code:
Structural v.s. Behavioral

// Structural specification

```
module  example1 (x1, x2, s, f);
    input  x1, x2, s;
    output  f;

    not (k, s);
    and (g, k, x1);
    and (h, s, x2);
    or (f, g, h);

endmodule
```

// Behavioral-Procedural specification

```
module  example5 (x1, x2, s, f);
    input  x1, x2, s;
    output f;
    reg f;

    always @(x1 or x2 or s)
        if (s == 0)
            f = x1;
        else
            f = x2;

endmodule
```

[Figure 2.37 from the textbook]

[Figure 2.42 from the textbook]

Behavioral Verilog:
Continuous v.s. Procedural

// Behavioral-Continuous specification

```verilog
module example3 (x1, x2, s, f);
    input x1, x2, s;
    output f;

    assign f = (~s & x1) | (s & x2);

endmodule
```

[Figure 2.40 from the textbook]

// Behavioral-Procedural specification

```verilog
module example5 (x1, x2, s, f);
    input x1, x2, s;
    output f;
    reg f;

    always @(x1 or x2 or s)
        if (s == 0)
            f = x1;
        else
            f = x2;

endmodule
```

[Figure 2.42 from the textbook]

Verilog Code

Structural
(basic components)

Behavioral

Continuous
(assign statements)

Procedural
(always statements)

Another Example

Let's Write the Code for This Circuit

[Figure 2.39 from the textbook]

Structural Verilog

module example2 (x1, x2, x3, x4, f, g, h);
      input x1, x2, x3, x4;
      output f, g, h;

      and (z1, x1, x3);
      and (z2, x2, x4);
      or (g, z1, z2);
      or (z3, x1, ~x3);
      or (z4, ~x2, x4);
      and (h, z3, z4);
      or (f, g, h);

endmodule

[Figure 2.39 from the textbook]   [Figure 2.38 from the textbook]

Structural Verilog

module example2 (x1, x2, x3, x4, f, g, h);
  input x1, x2, x3, x4;
  output f, g, h;

  and (z1, x1, x3);
  and (z2, x2, x4);
  or (g, z1, z2);
  or (z3, x1, ~x3);
  or (z4, ~x2, x4);
  and (h, z3, z4);
  or (f, g, h);

endmodule

[Figure 2.39 from the textbook]   [Figure 2.38 from the textbook]

Behavioral Verilog

module example4 (x1, x2, x3, x4, f, g, h);
    input x1, x2, x3, x4;
    output f, g, h;

    assign g = (x1 & x3) | (x2 & x4);
    assign h = (x1 | ~x3) & (~x2 | x4);
    assign f = g | h;

endmodule

[Figure 2.39 from the textbook]   [Figure 2.41 from the textbook]

Structural v.s. Behavioral

module example2 (x1, x2, x3, x4, f, g, h);
      input x1, x2, x3, x4;
      output f, g, h;

      and (z1, x1, x3);
      and (z2, x2, x4);
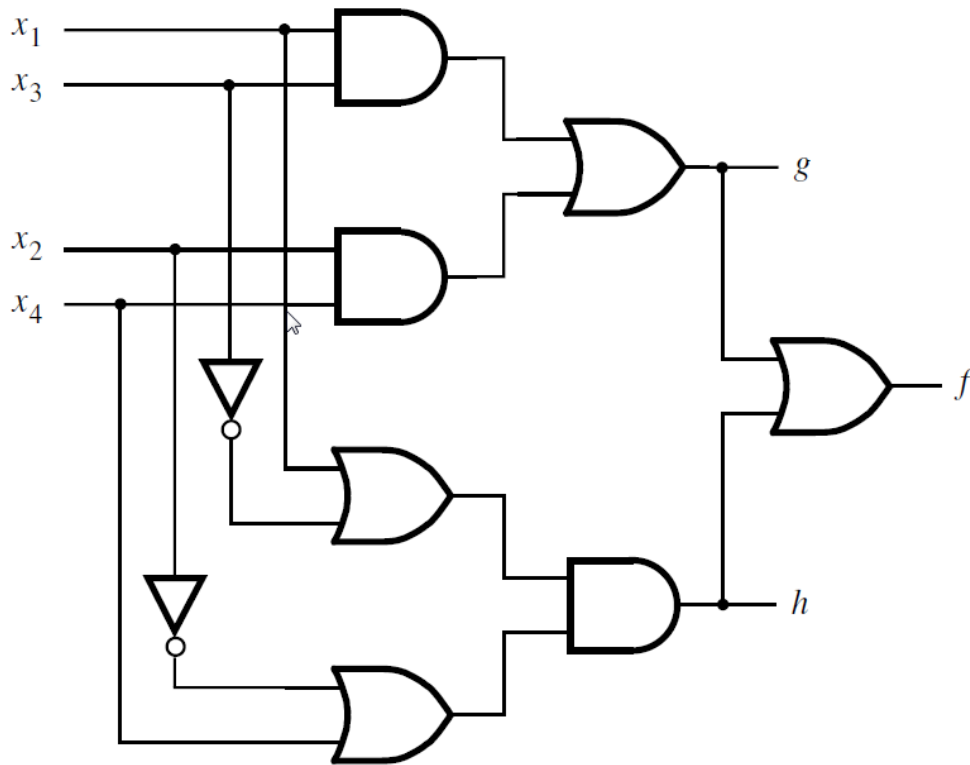   **or** (g, z1, z2);
      or (z3, x1, ~x3);
      or (z4, ~x2, x4);
      and (h, z3, z4);
      or (f, g, h);

endmodule

// structural

module example4 (x1, x2, x3, x4, f, g, h);
      input x1, x2, x3, x4;
      output f, g, h;

      assign g = (x1 & x3) | (x2 & x4);
      assign h = (x1 | ~x3) & (~x2 | x4);
      assign f = g | h;

endmodule

// behavioral-continuous

Yet Another Example

A logic circuit with two modules

[Figure 2.44 from the textbook]

The adder module

$$
\begin{array}{c}
a \\
+\,b \\
\hline
s_1\,s_0
\end{array}
\qquad
\begin{array}{c}
0 \\
+\,0 \\
\hline
0\;0
\end{array}
\qquad
\begin{array}{c}
0 \\
+\,1 \\
\hline
0\;1
\end{array}
\qquad
\begin{array}{c}
1 \\
+\,0 \\
\hline
0\;1
\end{array}
\qquad
\begin{array}{c}
1 \\
+\,1 \\
\hline
1\;0
\end{array}
$$
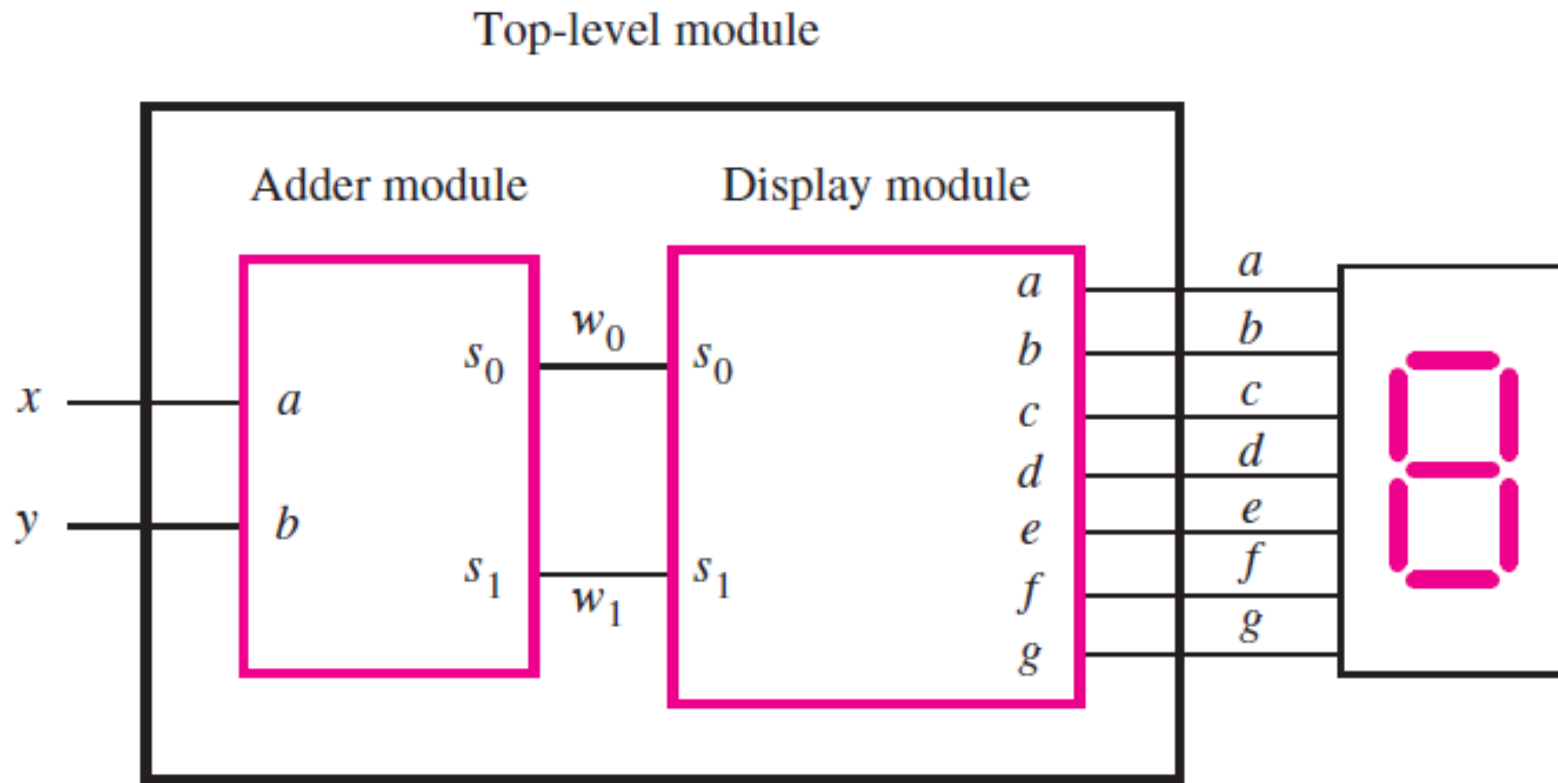
(a) Evaluation of $S = a + b$

a	b	s_1	s_0
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

(b) Truth table

(c) Logic network

The adder module
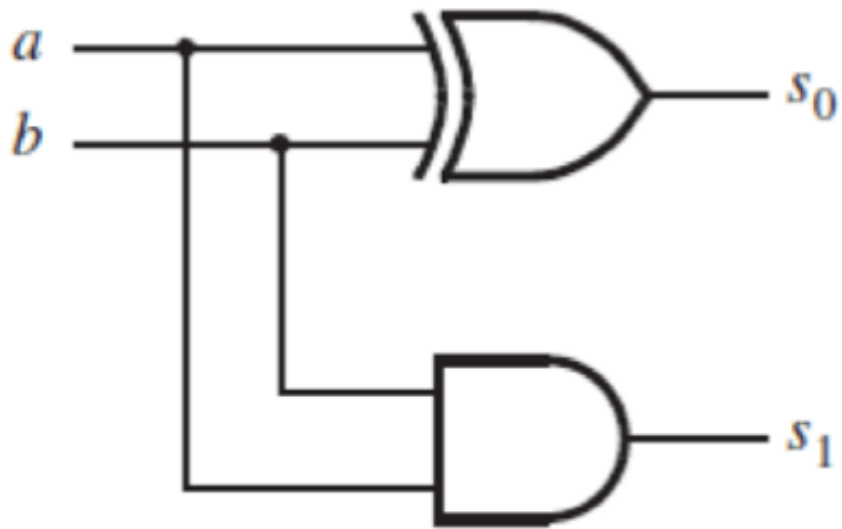

```verilog
// An adder module
module  adder (a, b, s1, s0);
    input a, b;
    output s1, s0;

    assign s1 = a & b;
    assign s0 = a ^ b;

endmodule
```

[Figure 2.45 from the textbook]

The display module

s_1	s_0	a	b	c	d	e	f	g
0	0	1	1	1	1	1	1	0
0	1	0	1	1	0	0	0	0
1	0	1	1	0	1	1	0	1

$$a = \overline{s_0} \qquad c = \overline{s_1} \qquad e = \overline{s_0} \qquad g = s_1\overline{s_0}$$

$$b = 1 \qquad d = \overline{s_0} \qquad f = \overline{s_1}\,\overline{s_0}$$

The display module

$$a = \overline{s_0}$$

$$b = 1$$

$$c = \overline{s_1}$$

$$d = \overline{s_0}$$

$$e = \overline{s_0}$$

$$f = \overline{s_1}\,\overline{s_0}$$

$$g = s_1\overline{s_0}$$

```
// A module for driving a 7-segment display
module display (s1, s0, a, b, c, d, e, f, g);
        input s1, s0;
        output a, b, c, d, e, f, g;

        assign a = ~s0;
        assign b = 1;
        assign c = ~s1;
        assign d = ~s0;
        assign e = ~s0;
        assign f = ~s1 & ~s0;
        assign g = s1 & ~s0;

endmodule
```

[Figure 2.46 from the textbook]

Putting it all together

Top-level module


```
// An adder module
module adder (a, b, s1, s0)
    input a, b;
    output s1, s0;

    assign s1 = a & b;
    assign s0 = a ^ b;

endmodule
```
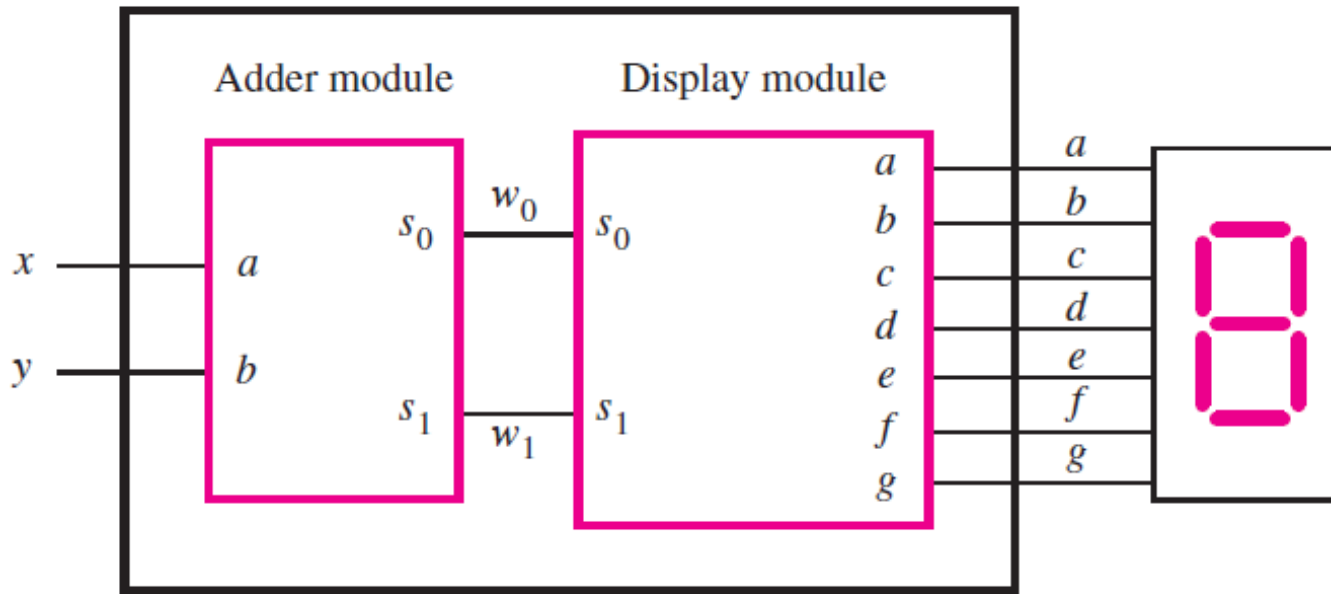
```
// A module for driving a 7-segment display
module display (s1, s0, a, b, c, d, e, f, g);
    input s1, s0;
    output a, b, c, d, e, f, g;

    assign a = ~s0;
    assign b = 1;
    assign c = ~s1;
    assign d = ~s0;
    assign e = ~s0;
    assign f = ~s1 & ~s0;
    assign g = s1 & ~s0;

endmodule
```
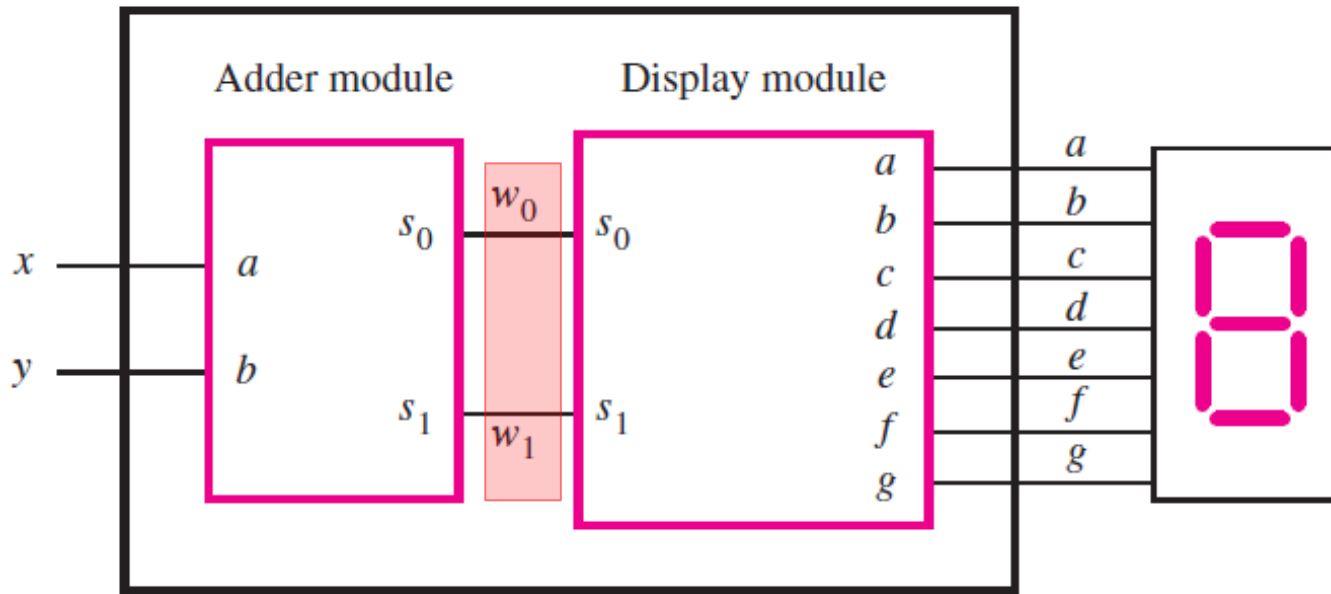
```
module adder_display (x, y, a, b, c, d, e, f, g);
    input x, y;
    output a, b, c, d, e, f, g;
    wire w1, w0;

    adder U1 (x, y, w1, w0);
    display U2 (w1, w0, a, b, c, d, e, f, g);

endmodule
```

Putting it all together

Top-level module


```
// An adder module
module  adder (a, b, s1, s0)
    input a, b;
    output s1, s0;

    assign s1 = a & b;
    assign s0 = a ^ b;

endmodule
```

```
// A module for driving a 7-segment display
module display (s1, s0, a, b, c, d, e, f, g);
    input s1, s0;
    output a, b, c, d, e, f, g;

    assign a = ~s0;
    assign b = 1;
    assign c = ~s1;
    assign d = ~s0;
    assign e = ~s0;
    assign f = ~s1 & ~s0;
    assign g = s1 & ~s0;

endmodule
```

```
module adder_display (x, y, a, b, c, d, e, f, g);
    input x, y;
    output a, b, c, d, e, f, g;
    wire w1, w0;

    adder U1 (x, y, w1, w0);
    display U2 (w1, w0, a, b, c, d, e, f, g);

endmodule
```

Questions?

THE END